目录 MATLAB 6.5

第 1 章　神经网络概述

人工神经网络特有的非线性适应性信息处理能力，克服了传统人工智能方法对于直觉，如模式、语音识别、非结构化信息处理方面的缺陷，使之在神经专家系统、模式识别、智能控制、组合优化、预测等领域得到成功应用。人工神经网络与其他传统方法相结合，将推动人工智能和信息处理技术不断发展。近年来，人工神经网络正向模拟人类认知的道路上更加深入发展，与模糊系统、遗传算法、进化机制等结合，形成计算智能，成为人工智能的一个重要方向，将在实际应用中得到发展。

用 MATLAB 语言构造典型神经网络的激活函数，编写各种网络设计与训练的子程序，网络的设计者可以根据自己的需要去调用工具箱中有关神经网络的设计训练程序，使自己能够从繁琐的编程中解脱出来。

本章主要对 MATLAB 语言和神经网络做简要的介绍，已经有这方面基础的读者可以跳过本章的学习。

本章主要内容包括：

- MATLAB 6.5 语言简介
- 神经网络发展史
- 神经网络模型
- 人工神经网络的特性

1.1　MATLAB 6.5 语言简介

MATLAB 语言是一种非常强大的工程语言，本节主要对该语言进行简单的介绍，包括 MATLAB 的产生背景及主要产品、MATLAB 的语言特点，以及 MATLAB 6.5 新版本所具有的新特点。

1.1.1　MATLAB 的产生背景及主要产品

MATLAB 诞生于 20 世纪 70 年代，它的编写者是 Cleve Moler 博士和他的同事。当时，Cleve Moler 博士与他的同事开发了 EISPACK 和 LINPACK 的 Fortran 子程序库。这两个程序库主要是求解线性方程的程序库。但是，Cleve Moler 发现学生使用这两个程序库时有困难，主要是接口程序不好写，很费时间。于是 Cleve Moler 自己动手，在业余时间里编写了 EISPACK 和 LINPACK 的接口程序。Cleve Moler 给这个接口程序取名为 MATLAB，意为矩阵（MATRIX）和实验室（LABORATORY）的组合。以后几年，MATLAB 作为免费软件在大学里使用，深受大学生的喜爱。

1984 年，Cleve Moler 和 John Little 成立了 MathWorks 公司，正式把 MATLAB 推向市

场，并继续进行 MATLAB 的开发。1993 年，MathWorks 公司推出 MATLAB 4.0；1995 年，MathWorks 公司推出 MATLAB4.2C 版（For Win3.x）；1997 年推出 MATLAB 5.0，2000 年 10 月，MathWorks 公司推出 MATLAB 6.0，2002 年 8 月，新的版本 MATLAB 6.5 已经开始发布了。每一次版本的推出都使 MATLAB 有长足的进步，界面越来越友好，内容越来越丰富，功能越来越强大。它的帮助信息采用超文本格式和 PDF 格式，可以很方便地阅读。

MATLAB 长于数值计算，能处理大量的数据，而且效率比较高。MathWorks 公司在此基础上开拓了符号计算、文字处理、可视化建模和实时控制能力，增强了 MATLAB 的市场竞争力，使 MATLAB 成为了市场主流的数值计算软件。

MATLAB 产品族是支持从概念设计、算法开发、建模仿真、实时实现的理想的集成环境。无论是进行科学研究和产品开发，MATLAB 产品族都是必不可少的工具。

MATLAB 产品族可用来进行：

- 数据分析
- 数值和符号计算
- 工程与科学绘图
- 控制系统设计
- 数字图像信号处理
- 财务工程
- 建模、仿真、原型开发
- 应用开发
- 图形用户界面设计

MATLAB 产品族被广泛地应用于包括信号与图像处理、控制系统设计、通信、系统仿真等诸多领域。开放式的结构使 MATLAB 产品族很容易针对特定的需求进行扩充，从而在不断深化对问题的认识的同时，提高自身的竞争力。

MATLAB 产品族的一大特性是有众多的面向具体应用的工具箱和仿真块，包含了完整的函数集用来对信号图像处理、控制系统设计、神经网络等特殊应用进行分析和设计。其他的产品延伸了 MATLAB 的能力，包括数据采集、报告生成和依靠 MATLAB 语言编程产生独立 C/C++代码等。

MATLAB 主要产品构成如下：

- MATLAB　所有 MathWorks 公司产品的数值分析和图形基础环境。MATLAB 将 2D 和 3D 图形、MATLAB 语言能力集成到一个单一的、易学易用的环境之中。
- MATLAB Toolbox　一系列专用的 MATLAB 函数库，解决特定领域的问题。工具箱是开放的可扩展的：可以查看其中的算法或开发自己的算法。
- MATLAB Compiler　将 MATLAB 语言编写的 M 文件自动转换成 C 或 C++文件，支持用户进行独立应用开发。结合 Mathworks 提供的 C/C++数学库和图形库，用户可以利用 MATLAB 快速地开发出功能强大的独立应用。
- Simulink　是结合了框图界面和交互仿真能力的非线性动态系统仿真工具。它以 MATLAB 的核心数学、图形和语言为基础。
- Stateflow　与 Simulink 框图模型相结合，描述复杂事件驱动系统的逻辑行为，驱

动系统在不同的模式之间进行切换。

- Real-Time Workshop 直接从 Simulink 框图自动生成 C 或 Ada 代码，用于快速原型和硬件在回路仿真，整个代码生成可以根据需要完全定制。
- Simulink Blockset 专门为特定领域设计的 Simulink 功能块的集合，用户也可以利用已有的块或自编写的 C 和 MATLAB 程序建立自己的块。

1.1.2 MATLAB 语言特点

MATLAB 语言有不同于其他高级语言的特点，它被称为第四代计算机语言。正如第三代计算机语言如 Fortran 语言与 C 语言等使人们摆脱了对计算机硬件的操作一样，MATLAB 语言使人们从繁琐的程序代码中解放出来。它的丰富的函数使开发者无需重复编程，只要简单地调用和使用即。MATLAB 语言最大的特点是简单和直接。MATLAB 语言的主要特点有以下几点。

1．编程效率高

MATLAB 是一种面向科学与工程计算的高级语言，允许用数学形式的语言编写程序，且比 Basic、Fortran 和 C 等语言更加接近我们书写计算公式的思维方式，用 MATLAB 编写程序犹如在演算纸上排列出公式与求解问题。因此，MATLAB 语言也可通俗地称为演算纸式科学算法语言。由于它编写简单，所以编程效率高，易学易懂。

2．用户使用方便

MATLAB 语言是一种解释执行的语言（在没有被专门的工具编译之前），它灵活、方便，其调试程序手段丰富，调试速度快，需要学习时间少。人们用任何一种语言编写程序和调试程序一般都要经过四个步骤：编辑、编译、连接及执行和调试。各个步骤之间是顺序关系，编程的过程就是在它们之间做瀑布型的循环。MATLAB 语言与其他语言相比，较好地解决了上述问题，把编辑、编译、连接和执行融为一体。它能在同一画面上进行灵活操作，快速排除输入程序中的书写错误、语法错误及语意错误，从而加快了用户编写、修改和调试程序的速度，可以说在编程和调试过程中它是一种比 VB 还要简单的语言。

具体地说，MATLAB 运行时，如直接在命令行输入 MATLAB 语句（命令），包括调用 M 文件的语句，每输入一条语句，就立即对其进行处理，完成编译、连接和运行的全过程。又如，将 MATLAB 源程序编辑为 M 文件，由于 MATLAB 磁盘文件也是 M 文件，所以编辑后的源文件就可直接运行，而不需进行编译和连接。在运行 M 文件时，如果有错，计算机屏幕上会给出详细的出错信息，用户经修改后再执行，直到正确为止。所以可以说，MATLAB 语言不仅是一种语言，而且从广义上讲是一种该语言开发系统，即语言调试系统。

3．扩充能力强，交互性好

高版本的 MATLAB 语言有丰富的库函数，在进行复杂的数学运算时可以直接调用，而且 MATLAB 的库函数与用户文件在形成上一样，所以用户文件也可作为 MATLAB 的库函数来调用。因而，用户可以根据自己的需要方便地建立和扩充新的库函数，以便提高 MATLAB 的使用效率和扩充它的功能。另外，为了充分利用 Fortran、C 等语言的资源，

包括用户已编好的 Fortran、C 语言程序，通过建立 Me 调文件的形式，混合编程，方便地调用有关的 Fortran、C 语言的子程序；还可以在 C 语言和 Fortran 语言中方便地使用 MATLAB 的数值计算功能，这样良好的交互性使程序员可以使用以前编写过的程序，减少重复性工作，也使现在编写的程序具有重复利用的价值。

4．移植性和开放性都很好

MATLAB 是用 C 语言编写的，而 C 语言的可移植性很好。于是 MATLAB 可以很方便地移植到能运行 C 语言的操作平台上。MATLAB 适合的工作平台有：Windows 系列、UNIX、Linux、VMS6.1、PowerMac。除了内部函数外，MATLAB 所有的核心文件和工具箱文件都是公开的，都是可读可写的源文件，用户可以通过对源文件的修改和自己编程构成新的工具箱。

5．语句简单，内涵丰富

MATLAB 语言中最基本最重要的成分是函数，其一般形式为［a，b，c，…］= fun（d，e，f，…），即一个函数由函数名、输入变量（d，e，f,…）和输出变量（a，b，c，…）组成。同一函数名 F，不同数目的输入变量（包括无输入变量）及不同数目的输出变量，代表着不同的含义（有点像面向对象中的多态性。这不仅使 MATLAB 的库函数功能更丰富，而且大大减小了需要的磁盘空间，使得 MATLAB 编写的 M 文件简单、短小而高效。

6．高效方便的矩阵和数组运算

MATLAB 语言像 Basic、Fortran 和 C 语言一样规定了矩阵的算术运算符、关系运算符、逻辑运算符、条件运算符及赋值运算符，而且这些运算符大部分都可以毫无改变地照搬到数组间的运算，有些如算术运算符只要增加“·”就可用于数组间的运算。另外，它不需定义数组的维数，并给出矩阵函数、特殊矩阵专门的库函数，使之在求解诸如信号处理、建模、系统识别、控制、优化等领域的问题时，显得大为简捷、高效、方便，这是其他高级语言所不能比拟的。在此基础上，高版本的 MATLAB 已逐步扩展到科学及工程计算的其他领域。因此，不久的将来，它一定能名副其实地成为“万能演算纸式的”科学算法语言。

7．方便的绘图功能

MATLAB 的绘图是十分方便的，它有一系列绘图函数（命令），例如线性坐标、对数坐标、半对数坐标及极坐标，均只需调用不同的绘图函数（命令），在图上标出图题、*XY* 轴标注，格（栅）绘制也只需调用相应的命令，简单易行。另外，在调用绘图函数时调整自变量可绘出不变颜色的点、线、复线或多重线。这种为科学研究着想的设计是通用的编程语言所不及的。

1.1.3 MATLAB 6.5 的新特点

2002 年，MathWorks 公司发布 MATLAB 最新 6.5 版产品。MATLAB 6.5 的特点在于全新的桌面及各种不同领域的集成工具，使用户易于使用。多种新工具简化了一般的工作如资料输入、快速分析，并创造高品质且具实用性的图表分析等。MATLAB 6.5 包含了新

的 JIT 加速度计，JIT 加速度计有力地增加了 MATLAB 中的许多操作和数据类型的计算速度。其他新的特色和加强包括以下三个方面。

1. 编程和数据类型

（1）增加了变量名、函数名和文件名的最大长度：变量名、函数名、子函数名、结构域名、M 文件名、MEX 文件名和 MDL 文件名可以达到 63 个字节；

（2）支持 64 位的文件偏移量，能够为大于 2 GB 的数据文件实现低层次的 I/O 函数；

（3）支持有符号和无符号的 64 位整数；

（4）支持使用动态域名来访问和修改结构数组；

（5）简化了 AND 和 OR 逻辑运算；

（6）支持新的 MATLAB 定时器，而不是定时执行 MATLAB 命令；

（7）改进了音频支持；

（8）加强了警告和错误提示功能：新支持格式化的字符串和消息标识符。

2. 外部接口

（1）改进了自动化客户接口：新的查看和修改属性用户接口，增强了事件和例外句柄；

（2）增强了网络集成：读 URL 的内容，在 MATLAB 中发送 E-mail，以及解压缩文件。

3. 开发环境

（1）新的 M-文件接口，能更好地理解 M-代码；

（2）新的启动按钮，易于执行共同的命令；

（3）改进了文件和目录管理工具；

（4）增强了数组编辑器：与 Excel 之间剪切、复制、删除和交换单元的新功能，支持更大的数组；

（5）改进了编辑和调试工具；

（6）改进与 PC 平台的控制接口；

（7）支持新的图形用户接口，从 HDF 或 HDF-EOS 文件导入数据；

（8）JVM1.3.1 支持 Windows，Linox 和 Solaris 平台。

4. 图形

提高了图形性能：新的彩色图形编辑器；改进了图形特性编辑器。

5. 数学

（1）新的数学计算和算法改进；

（2）在 Pentium4 上更快地计算许多函数，比如矩阵乘法、矩阵转置和线性代数运算；

特别指出的是，许多新特色并不是适合于所有的平台。

1.2 神经网络发展史

人工神经元网络的研究已有近半个世纪的历史，但它的发展并不是一帆风顺的，而是经过两起一落中间呈现马鞍形的过程。下面以时间的顺序和著名人物或者某方面的突出研究成果为线索，简要地介绍其发展史。它的研究大体上可分为四个阶段。

1．早期阶段

人工神经系统的研究可以追溯到 1800 年 Frued 的前精神分析学时期，他已经做了一些初步工作。1913 年人工神经系统的第一个实践是由 Russell 描述的水力装置。1943 年美国心理学家 Warren S McCulloch 与数学家 Water H Pitts 合作，用逻辑的数学工具研究客观事件在形式神经网络中的描述，从此开创了对神经网络的理论研究。他们在分析、总结神经元基本特性的基础上，首先提出了神经元的数学模型，简称为 MP 模型。从脑科学研究来看，MP 模型不愧为第一个用数理语言描述脑的信息处理过程的模型。后来 MP 模型经过数学家的精心整理和抽象，最终发展成一种有限自动机理论，再一次展现了 MP 模型的价值。此模型沿用至今，直接影响着这一领域研究的进展。

1949 年心理学家 D.O.Hebb 提出了关于神经网络学习机理的“突触修正假设”，即突触联系效率可变的假设。Hebb 学习规则开始是作为假设提出来的，其正确性在 30 年后才得到证实，现在多数学习机仍遵循这一规律。

1957 年 F.Rosenblatt 首次提出并设计制作了著名的感知器（Perceptron），第一次从理论研究转入工程实现阶段，掀起了研究人工神经元网络的高潮。

1962 年 Bernard Widrow 和 Marcian Hoff 提出了自适应线性元件网络，简称为 Adaline(Adaptive linear element)。它是一种连续取值的线性加权求和阈值网络，它也可以看成为感知器的变形，它实质上是一个两层前馈感知机型网络。它成功地应用于自适应信号处理和雷达天线控制等连续可调过程。

2．20 世纪 70 年代的过渡

进入 20 世纪 70 年代后，虽然神经网络相对处于低潮时期，但是仍有不少科学家在极其困难的条件下坚持不懈地努力奋斗，主要是提出了各种不同的网络模型，开展了人工神经网络理论、增加网络的功能和各种学习算法的研究等，为今后研究神经网络理论、数学模型和体系结构等方面打下了坚实的基础。

Stephen Grossberg 是所有研究人工神经系统人员中最有影响者，他周到、广泛地研究了心理学（思维）和生物学（脑）的处理，以及人类信息处理的现象，把思维及脑紧密结合在一起，成为统一的理论。

日本学者 Shun-Ichi Amari 致力于神经网络有关数学理论的研究。他的突出结果是 1971 年的对称连接人工神经系统的稳定性研究与 1979 年和 Kishimoto 合作发表的在不对称连接人工神经系统中回忆瞬时模式序列的稳定性的研究。在 1982 年和 1983 年还发表了人工神经系统中模式结构的动力学。

1970 年和 1973 年，Kunihiko Fukushima 研究了视觉系统的空间和时空的人工神经系统模型，以及脑的空间和时空的人工系统模型。提出了神经认知网络理论。Fukushima

网络包括人工神经认知和基于人工神经认知机的有选择注意力的识别两个模型。

1971 年，芬兰的 Tuevo Kohonen 开始从事随机连接变化表的研究工作。从 1972 年开始，他很快集中到联想记忆（相关矩阵）方面。最近，Kohonen 已经将 LVQ 应用到语音识别、模式识别和图像识别上面。Kohonen 的工作已经集中到联想记忆的三个专题：系统理论方法、相联存储器（内容可寻址）和自组织及联想记忆。

1968 年 James Anderson 从具有基于神经元突触的激活联想记忆模型的 ANS 模型开始工作。1973 年和 1977 年他把线性联想记忆（LAM）应用到诸如识别、重构和任意可视模式的联想这样的问题上。1977 年，Anderson，Silvetstein 等人在 LAM 工作上又有了重要进展，通过加正水平反馈，使用误差修正学习和用斜坡函数代替阈值函数，建立起称为 BSB（Brain-state-in-a-box）的模型。在 20 世纪 80 年代中期，BSB ANS 已经被用来解释概念形成、分类和知识处理。

1979 年日本 NHK 的福岛邦彦（Fukushima K）提出了认知机模型，后来又提出了改进型认知机（Neocognition）模型。

1979 年日本东京大学的中野馨提出了有名的联想记忆模型，即所谓联想机（Associatron），它能实现从残余信息（模式）到完整信息（模式）的恢复过程。

3．20 世纪 80 年代的新高潮

神经网络研究第二次高潮到来的标志和揭开神经网络计算机研制序幕的是美国加州工学院物理学家 John Hopfield，他于 1982 年和 1984 年在美国科学院院刊上发表了两篇文章，提出了模仿人脑的神经网络模型，即著名的 Hopfield 模型。Hopfield 网络是一个互连的非线性动力学网络，它解决问题的方法是一种反复运算的动态过程，这是符号逻辑处理方法所不具备的性质。

Jerome Feldman 和 Dana Ballard 1980 年在 Rochester 大学是最先从事人工神经系统研究人员中的两人。他们已经开发出许多不同的人工神经系统，其早期工作在视觉方面，此外，这个组还研究了自然语言、语义网络、逻辑推理和概念表示等，还提出了连接结构网络模型，并指出了传统人工智能计算与生物计算的区别，以及并行分布式处理的计算原则。

Terrence Sejnowski 于 1976 年开始从事人工神经元系统方面的工作，他是少数研究人员中具有强数学和生物背景的一位研究人员。他的第一项工作集中在寻找对于协方差学习规则的神经逻辑学上的证明。1984 年和 1985 年他与 Hinton，Ackley 用统计物理学的概念和方法研究神经网络，提出了 Boltzmann（玻尔兹曼）机。

McClelland 和 Rumelhart 是认知心理学家。他们最初感兴趣的是：用人工神经系统模型去帮助理解思维的心理学功能。1981 年和 1982 年经过他们的共同努力，设计了交互激活模型的第一个 ANS 变化表，它被用来解释词的识别。后来，推广到其他的 ANS 结构，并用于并行分布式处理（Parallel Distributed Processing，PDP）来描述他们的工作，引起了许多其他研究人员的兴趣。1986 年他们提出了多层网络的误差反传算法（Back Propagation，简称为 BP 算法）。

4．20 世纪 80 年代后期以来的热潮

1987 年 6 月 21 日在美国圣地亚哥召开了第一届国际神经网络学术会议，宣告了国际神经网络协会正式成立，与会者 1600 多人，会上不但宣告了神经网络计算机学科的诞生，

而且还展示了有关公司、大学所开发的神经网络计算机方面的产品和芯片，掀起了人类向生物学习、研究和开发及应用神经网络的新热潮。在这之后，每年都要召开神经网络和神经计算机的国际性和地区性会议，促进神经网络的研制、开发和应用。

1991 年 IJCNN（国际联合神经网络会议）会议主席 D.Rumelhart 在开幕词中讲到“神经网络的发展已进入转折点，它的范围正在不断扩大，领域几乎包括各个方面”。INNS（国际神经网络学会）主席 P.Werbos 指出：过去几年至过去几个月，神经网络的应用使工业技术发生了很大变化，特别是在控制领域有突破性进展。

这段时期以来，神经网络理论的应用取得了令人瞩目的进展，特别是在人工智能、自动控制、计算机科学、信息处理、机器人、模式识别、CAD/CAM 等方面都有重大的应用实例。下面列出一些主要的应用领域：

- 模式识别和图像处理　印刷体和手写体字符识别、语音识别、签字识别、指纹识别、人脸识别、人体病理分析、目标检测与识别、图像压缩和图像复原等。
- 控制和优化　化工过程控制、机器人运动控制、家电控制、半导体生产中掺杂控制、石油精炼优化控制和超大规模集成电路布线设计等。
- 预报和智能信息管理　股票市场预测、地震预报、有价证券管理、借贷风险分析、IC 卡管理和交通管理。
- 通信　自适应均衡、回波抵消、路由选择和 ATM 网络中的呼叫接纳识别及控制等。
- 空间科学　空间交会对接控制、导航信息管理智能管理、飞行器制导和飞行程序优化管理等。

神经网络与专家系统相结合已成为重要的发展趋势，这两者的结合能更好地发挥出各自的专长。1990 年 4 月 IBM 公司推出 AS400 工作站，其中就提供了一个自由的神经网络仿真开发环境。1991 年 5 月 SGI 公司宣布今后它的所有机器都提供 Accurate Automation 软件开发环境，目前该公司已利用这一环境进行航天飞机控制臂的神经网络控制设计，1992 年上半年进行实际飞行试验。此外，新的软件包、加速板、新的芯片和新方法不断深入研究，使 ANN 应用在速度和存储能力方面得到改进。

目前各国发展的重点是以应用为导向，以发展更高性能的混合计算机为目标。这些计划是以长远发展目标与近期效果相结合的，充分考虑到了与当前发展技术水平相适应。

在最近几年里，我国在人工神经元网络的研究方面发展规模大、速度快，而且取得了不少成果。1988 年由北京大学组织召开了第一次关于神经网络的讨论会，一些国际知名学者在会上做了专题报告。1989 年和 1990 年，不同学会和研究单位召开过专题讨论会。在 1990 年 12 月由 8 个单位联合发起和组织了中国第一次神经网络会议，IEEE 神经网络委员会副主席在会上做了神经网络主要动向的演说。1991 年由 13 个单位发起和组织召开了第二次中国神经网络会议。1992 年中国神经网络委员会在北京承办了世界性的国际神经网络学术大会，这届大会受到 IEEE 神经网络委员会、国际神经网络学会等国际学术组织的大力支持。这标志着我国神经网络的研究工作者第一次大规模地走向世界，这必将会进一步推动我国的神经网络研究。

1.3 神经网络模型

神经网络的基本单元是神经元，它是由大量的神经元广泛互连而成的网络，所以我们先从神经元谈起。

1.3.1 生物神经元模型

神经元是脑组织的基本单元，其结构如图 1-1 所示，神经元由三部分构成：细胞体、树突和轴突；每一部分虽具有各自的功能，但相互之间是互补的。

树突是细胞的输入端，通过细胞体间连接的节点“突触”接收四周细胞传出的神经冲动；轴突相当于细胞的输出端，其端部的众多神经末梢为信号的输出端子，用于传出神经冲动。

神经元具有兴奋和抑制的两种工作状态。当传入的神经冲动，使细胞膜电位升高到阈值（约为 40mV）时，细胞进入兴奋状态，产生神经冲动，由轴突输出。相反，若传入的神经冲动使细胞膜电位下降到低于阈值时，细胞进入抑制状态，没有神经冲动输出。

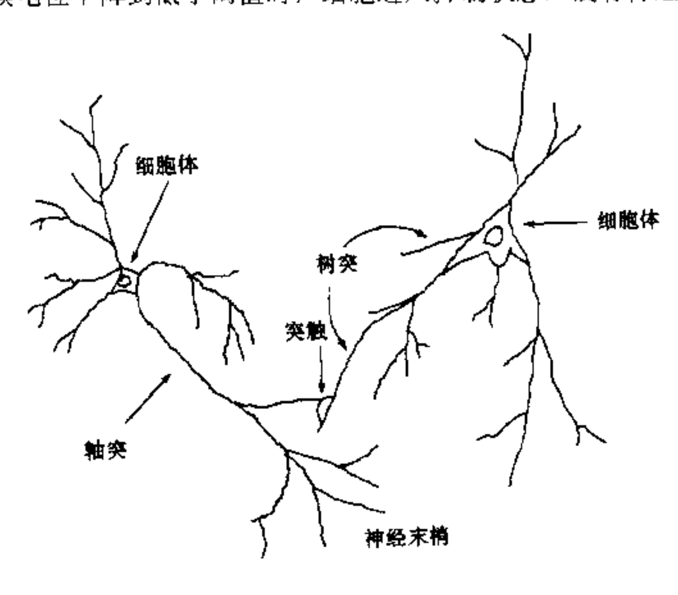


图 1-1 神经元结构

1.3.2 神经元模型

连接机制结构的基本处理单元与神经生理学类比往往称为神经元，每个构造起网络的神经元模型模拟一个生物神经元。人工神经元是对生物神经元的简化和模拟，它是神经网络的基本处理单元。图 1-2 给出了一种简化的神经元结构。它是一个多输入、单输出的非线性元件，其输入输出关系可描述为：

$$I_i = \sum_{j=1}^{n} w_{ji} x_j - \theta_i$$

$$y_i = f(I_i)$$

其中 $x_j(j=1,2,\ldots,n)$ 是从其他细胞传来的输入信号，θ_i 为神经元单元的偏置（阈值），w_{ji} 表示从细胞 j 到细胞 i 的连接权值（对于激发状态，w_{ij} 取正值；对于抑制状态，w_{ij} 取负值），n 为输入信号数目。y_j 为神经元输出，t 为时间，$f(.)$ 称为传递函数，有时叫做激发或激励函数，往往采用 0 和 1 二值函数或 S 型函数，见图 1-2，这三种函数都是连续和非线性的。

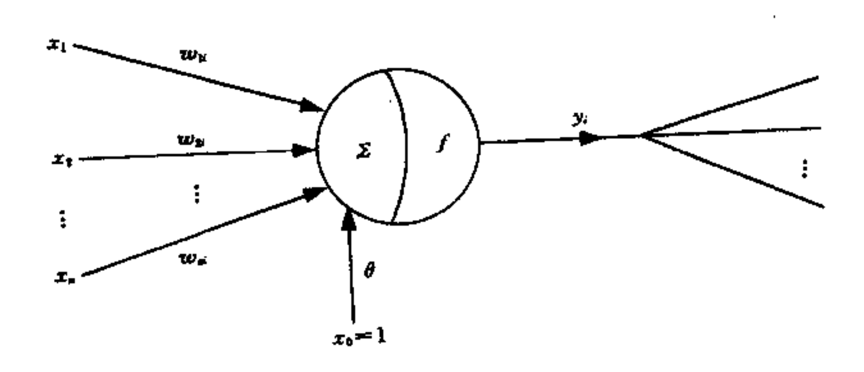


图 1-2　神经元结构模型

传递函数可为线性函数，但通常为像阶跃函数或 S 状曲线那样的非线性函数。常用的神经元非线性函数列举如下：

（1）阈值型函数　当 y_i 取 0 或 1 时，$f(x)$ 为阶跃函数：

$$f(x) = \begin{cases} 1 & x \geqslant 0 \\ 0 & x < 0 \end{cases}$$

（2）S 状曲线　通常是在（0，1）或（−1，1）内连续取值的单调可微分的函数，常用指数或正切等一类 S 状曲线来表示，如：

$$f(x) = \frac{1}{1+\exp(-\beta x)} \qquad (\beta > 0)$$

或

$$f(x) = \tanh(x)$$

当趋于无穷时，S 状曲线趋近于阶跃函数，通常情况下 β 取值为 1。

有时在网络中还采用下列计算简单的非线性函数：

$$f(x)=\frac{x}{1+|x|}$$

在 RBF(Radial Basis Function)构成的神经网络中，神经元的结构可用高斯函数描述如下：

$$y_i=\exp\left[-\frac{1}{2\sigma_i^2}\sum_j(x_j-w_{ji})^2\right]$$

这里 σ_i^2 为标准化参数。

有时为方便起见，常把 $-\theta_i$ 也看做是对应恒等于 1 的输入量 x_0 的权值，这时式中的和式可记为：

$$I_i=\sum_{j=0}^{n}w_{ji}x_j$$

其中 $$w_{0i}=-\theta_i,x_0=1$$

1.3.3 神经网络模型

神经网络是由大量的神经元广泛互连而成的网络。根据连接方式的不同，神经网络可分成两大类：没有反馈的前向网络和相互结合性网络。前向网络由输入层、中间层（或叫隐层）和输出层组成，中间层可有若干层，每一层的神经元只接受前一层神经元的输出。而相互连接型网络中任意两个神经元间都有可能连接，因此输入信号要在神经元之间反复往返传递，从某一初态开始，经过若干次的变化，渐渐趋于某一稳定状态或进入周期振荡等其他状态。

目前虽然已有数十种的神经网络模型，但已有的神经网络可分成三大类，即前向网络（Feedforward NNs）、反馈网络（Feedback NNs）和自组织网络（Self-organizing NNs）。下面将分别介绍这三类网络中有代表性的网络。

1.4 人工神经网络的特性

人工神经网络具有如下特性。

1．并行分布处理

神经网络具有高度的并行结构和并行实现能力，因而能够有较好的耐故障能力和较快

的总体处理能力。这特别适用于实时控制和动态控制。

2．非线性映射

神经网络具有固有的非线性特性，这源于其近似任意非线性映（变换）能力。这一特性给非线性控制问题带来新的希望。

3．通过训练进行学习

神经网络是通过研究系统过去的数据记录进行训练的。一个经过适当训练的神经网络具有归纳全部数据的能力。因此，神经网络能够解决那些由数学模型或描述规则难以处理的控制过程问题。

4．适应与集成

神经网络能够适应在线运行，并能同时进行定量和定性操作。神经网络的强适应和信息熔合能力使得网络过程可以同时输入大量不同的控制信号，解决输入信息间的互补和冗余问题，并实现信息集成和融合处理。这些特性特别适于复杂、大规模和多变量系统的控制。

5．硬件实现

神经网络不仅能够通过软件而且可借助软件实现并行处理。近年来，一些超大规模集成电路实现硬件已经问世，而且可从市场上购到。这使得神经网络具有快速和大规模处理能力的实现网络。

第2章 神经网络工具箱函数及实例

现在，使用神经网络能解决许多以前用传统方法无法解决的问题。神经网络在很多领域中都有应用，以实现各种复杂的功能。这些领域包括商业及经济估算、自动检测和监视、计算机视觉、语音处理、机器人及自动控制、优化问题、航空航天、银行金融业、工业生产等。而神经网络是一门发展很快的学科，其应用领域也会随着其发展有更大的拓宽。本章将讨论一些神经网络应用的实例。

使用神经网络的主要优点是能够自适应样本数据，当数据中有噪声、形变和非线性时，它也能够正常地工作；很容易继承现有的领域知识；使用灵活，能够处理来自多个资源和决策系统的数据；提供简单工具进行自动特征选取，产生有用的数据表示，可作为专家系统的前端（预处理器）。此外，神经网络还能提供十分快的优化过程，尤其以硬件直接实现网络时，而且可以加速联机应用程序的运行速度。当然，过分夸大神经网络的应用能力也是不恰当的，毕竟它不是无所不能的。这就需要在实际工作中具体分析问题，合理选择。

在 MATLAB 6.5 神经网络工具箱中提供了丰富的演示实例。本章的部分实例就来源于其中，并对其进行了更多的讨论和推广，有兴趣的读者可以在 MATLAB 命令行中输入 help nndemo，以得到神经网络的演示或者应用列表。

本章主要内容包括：

- 神经网络工具箱概述
- 感知器的神经网络工具函数
- BP 网络的神经网络工具函数
- 线性网络的神经网络工具函数
- 自组织竞争网络工具箱函数
- 径向基网络神经网络工具箱函数
- 回归网络的神经网络工具箱函数

2.1 神经网络工具箱概述

随着 MATLAB 软件的版本提高，其对应的神经网络工具箱的版本也在相应地提高。与 MATLAB 6.5 对应的神经网络工具箱为 NN Toolbox 4.0.2 版（以后简称神经网络工具箱），其内容非常丰富，包括了很多现有的神经网络的新成果，涉及的网络模型有：

- 感知器模型
- 线性滤波器
- BP 网络
- 控制系统网络模型
- 径向基网络

- 自组织网络
- 反馈网络
- 自适应滤波和自适应训练

神经网络工具箱提供了很多经典的学习算法，使用它能够快速地实现对实际问题的建模求解。由于其编程简单，这样就给使用者节省了大量的编程时间，使其能够把更多的精力投入到网络设计而不是具体程序实现上。

2.2 感知器的神经网络工具函数

MATLAB 提供了大量的感知器工具函数，本节将对这些函数的功能、调用格式，以及使用方法做详细的介绍。

2.2.1 MATLAB 6.5 中有关感知器的工具函数

MATLAB 6.5 提供了许多进行神经网络设计和分析的工具函数，这给用户带来了极大的方便。有关这些工具函数的使用可以通过 help 命令得到。表 2-1 列出了 MATLAB 6.5 中与感知器相关的神经网络重要工具函数。

表 2-1 感知器网络的重要函数和功能

函数名称	功能
newp	生成一个感知器神经网络
hardlim	硬限幅传递函数
hardlims	对称硬限幅传递函数
dotprod	权值点积函数
netsum	网络输入求和函数
initlay	某层的初始化函数
initwb	某层的权值和阈值的初始化函数
inizero	零权值阈值初始化函数
init	一个网络的初始化函数
mae	求平均绝对误差性能函数
learnp	感知器的学习函数
learnpn	标准感知器的学习函数
adaptwb	网络的权值阈值的自适应函数
adaptwb	神经网络的自适应
trainwb	网络的权值和阈值训练函数
train	神经网络训练函数
sim	神经网络仿真函数

2.2.2 工具函数详解

工具函数主要包括神经网络函数、权值函数、网络的输入函数、传递函数、初始化函数、性能函数、学习函数、自适应函数，以及训练函数等。

1. 神经网络函数

（1）newp

功能 生成一个感知器神经网络。

格式 net=newp(pr，s，tf，lf)。

说明 感知器适用于解决简单的（即线性可分的）分类问题。newp 的输入参数为：

- pr *R* 个输入向量的最大值和最小值的取值范围
- s 神经元的个数
- tf 网络的传递函数，默认值为 hardlim
- lf 网络的学习函数，默认值为 learnp

该函数执行之后返回一个新的感知器网络。传递函数 tf 可以选取 hardlim 或 hardlims 函数。学习函数 lf 可以选取 learnp 或 learnpn 函数。可以调用不带参数的 newp 函数在一个对话框中定义网络的属性。

参见 sim，init，adapt，train，hardlim，hardlims，learnp，learnpn。

（2）sim

功能 网络仿真。

格式 [Y，Pf，Af]=sim(net，P，Pi，Ai)

[Y，Pf，Af]=sim(net，{Q TS}，Pi，Ai)

[Y，Pf，Af]=sim(net，Q，Pi，Ai)

说明 sim 可仿真一个神经网络。sim 的参数为：

- net 神经网络
- P 网络的输入
- Pi 初始输入延迟，默认值为 0
- Ai 初始的层延迟，默认值为 0

且该函数返回：

- Y 网络的输出
- Pf 最终输出延迟
- Af 最终的层延迟



参数 Pi，Ai，Pf，Af 是可任选的，并且对于具有输入延迟或层的延迟的网络是必须使用的。

参见 sldebug，simset。

（3）init

功能 初始化神经网络。

格式 net=init(net)。

说明 使用 init(net)函数可以得到一个神经网络 net，该网络的权值和阈值是按照网络初始化函数来进行修正的。而网络的初始化函数是由 NET.initFcn 设定的，其参数是由 NET.initParam 确定的。

参见 initlay，initnw，initwb。

（4）adaptwb

功能 神经网络的自适应。

格式 [net，Y，E，Pf，Af]=adapt(NET，P，T，Pi，Ai)。

说明 函数 adaptwb 的各个参数的含义为：

- NET 一个神经网络
- P 网络的输入
- Pi 初始输入延迟，默认值为 0
- Ai 初始的层延迟，默认值为 0

且返回一个具有适应功能 NET.adaptFcn 及适应参数 NET.adaptParam 的结果：

- net 修正后的网络
- Y 网络的输出
- E 网络的误差
- Pf 最终输出延迟
- Af 最终的层延迟



参数 T 仅对需要目标的网络是必需的，而且是可任选的。Pi 及 Pf 仅用于具有输入或层间的延迟的网络，而且也是可任选的。

参见 sim，init，train。

（5）train

功能 神经网络的训练函数。

格式 [net，tr]=train(NET，P，T，Pi，Ai)

[net，tr]=train(NET，P，T，Pi，Ai，VV，TV)

说明 train 函数是按照 NET.trainFcn 和 NET.trainParam 训练网络 NET 的。Train 函数各个参数的含义为：

- NET 神经网络
- P 网络的输入
- T 网络的目标，默认值为 0
- Pi 初始输入延迟，默认值为 0
- Ai 初始的层延迟，默认值为 0

返回：

- net 修正后的网络
- tr 训练的记录（训练步数和性能 epoch and perf）



参数 T 仅对需要目标的网络是必需的，而且是可任选的. Pi 及 Pf 仅用于具有输入或层间的延迟的网络，而且也是可任选的。

2. 权值函数

dotprod

功能 权值点积函数。

格式 Z=dotprod(W，P)

df=dotprod('deriv')

说明 dotprod 是一个点积权值的函数，权值与输入的点积可得到加权输入。dotprod 函数的输入参数为：

- W S×R 维的权值矩阵
- P Q 组 R 维输入向量

返回 Q 组 S 维的 W 与 P 的点积。

参见 ddotprod，dist，negdist，normprod。

3. 网络的输入函数

netsum

功能 神经网络的输入求和函数。

格式 N=netsum(Z1，Z2，…)

df=netsum('deriv')

说明 netsum 是一个神经网络的输入求和函数。该函数的功能时间某一层的加权输入和阈值相加作为该层网络的输入。其中 netsum(Z1，Z2，…)可以取任意数目的输入，Zi 是 Q 组 S 维的矩阵，并且返回对 Zi 按元素求和的结果 N。netsum('deriv')函数返回 netsum 的导数函数。

参见 dnetsum，netprod，concur。

4. 传递函数

(1) hardlim

功能 硬限幅传递函数。

格式 a=hardlim(N)

info=hardlim(code)

说明 hardlim 是一个传递函数，它通过计算网络的输入得到该层的输出。如果网络输入达到门限，则硬限幅传递函数输出为 1，否则输出为 0。这表明神经网络可用于判断和分类。Hardlim 函数各参数的含义为：

N Q 组 S 维的网络输入向量，并返回该层的输出矩阵向量。当 N 大于 0 时，返回的元素是 1；当 N 小于 0 时，返回的元素是 0

Hardlim(code)与 hardlim(N)不同的是根据不同代码 code 返回有用的信息：

- 'deriv' 导数函数名
- 'name' 传递函数的全称

- 'output'　传递函数的输出范围
- 'activ'　传递函数的输入范围

参见　sim，hardlims。

（2）hardlims

功能　对称的硬限幅传递函数。

格式　a=hardlims(N)

　　　　info=hardlims(code)

说明　hardlims 是一个传递函数，它通过计算网络的输入得到该层的输出。如果网络输入达到门限，硬限幅传递函数输出为 1，否则输出为－1。这表明神经网络可用于判断和分类。Hardlim 函数各参数的含义为：

- N　N-Q 组 S 维的网络输入向量矩阵，并返回该层的输出矩阵向量。当 N 大于 0，返回的元素是 1；当 N 小于 0 时，返回的元素是－1

参见　hardlim。

5. 初始化函数

（1）initlay

功能　神经网络某一层的初始化函数。

格式　net=initlay(NET)

　　　　info=initlay(code)

说明　initlay 是一个网络的初始化函数，它是根据 NET.layers{I}.initFcn 设置的初始化函数对第 I 层的权进行初始化的。Initlay 函数各参数的含义为：

- NET　是一个神经网络
- Net　该函数返回一个第 I 层更新了的网络
- Initlay(code)　函数可按照 code 指定的代码返回一些有用的信息，其中 code 可取为：
  - 'pnames'　初始化参数的名称
  - 'pdefaults'　默认的初始化参数
  - initlay　函数没有任何初始化参数

参见　initwb，initnw，init。

（2）initwb

功能　神经网络的某一层的权值和阈值的初始化函数。

格式　net=initwb(net,i)。

说明　initwb 函数可以按照其特定的初始化函数对网络的某一层的权值和阈值进行初始化。

Initlay(net,i)的输入中 net 代表的是一个神经网络，i 表示网络的第 i 层，该函数返回一个第 *i* 层的权值和阈值都更新了的网络。

参见　initnw，initlay，init。

（3）initzero

功能　将权值设置为零的初始化函数。

格式　W=initzero(S，RR)

b=initzero(S，[1 1])

说明　initzero(S，RR)函数是将权值设置为零的初始化函数，其中包含神经元个数 S，即输入向量的范围 RR。该函数返回一个零权值矩阵。

initzero(S,[1 1])函数是将阈值设置为零的初始化函数，它能返回一个为 0 阈值向量。

参见　initwb，init，initlay。

6. 性能函数

mae

功能　平均绝对误差性能函数。

格式　perf=mae(e，x，pp)

perf=mae(e，net，pp)

info=mae(code)

说明　mae 是一个求网络的性能参数的函数。mae(e，x，pp)函数可采用 1～3 个参数：

- e　误差向量矩阵或向量
- x　所有的权值和阈值向量（可忽略）
- pp　性能参数（可忽略）

该函数执行后可返回平均绝对误差。误差 e 可以是元素组或矩阵。

参见　mse，msereg，dmae。

7. 学习函数

（1）learnp

功能　感知器的权值/阈值学习函数。

格式　[dW，LS]=learnp(W，P，Z，N，A，T，E，gW，gA，D，LP，LS)

[db，LS]=learnp(b，ones(1，Q)，Z，N，A，T，E，gW，gA，D，LP，LS)

info=learnp(code)

说明　learnp 函数是感知器的权值/阈值学习函数，其各个参数含义如下：

- W　S×R 维的权值矩阵（或 S×1 维的阈值向量）
- P　Q 组 R 维的输入向量（或 Q 组单个输入）
- Z　Q 组 S 维的权值输入向量
- N　Q 组 S 维的网络输入向量
- A　Q 组 S 维的输出向量
- T　Q 组 S 维的目标向量
- E　Q 组 S 维的误差向量
- gW　S×R 维的性能参数的梯度
- gA　Q 组 S 维的性能参数的输出梯度
- LP　学习参数，若没有学习参数，则 LP=[ ]
- LS　学习状态，初始值为[ ]

该函数可返回以下参数：

- dW　S×R 维的权值（或阈值）变化阵。

- LS 新的学习状态。
- Learnp(code) 对于每一个 code 代码返回相应的有用的信息：
  - 'pnames' 返回学习参数的名称
  - 'pdefaults' 返回默认的学习参数
  - 'needg' 如果该函数使用 gW 或 gA，则返回值为 1

参见 learnpn，newp，adaptwb。

（2）learnpn

功能 标准化感知器的权值/阈值学习函数。

格式 [dW，LS]=learnpn(W，P，Z，N，A，T，E，gW，gA，D，LP，LS)

info=learnpn(code)

说明 learnpn 函数是标准化感知器的权值/阈值学习函数。当输入样本中具有奇异的向量时，该学习函数能够产生较快的学习速度。其各个参数含义同 learnp。

参见 learnp。

8. 自适应函数

adaptwb

功能 网络的权值和阈值的自适应函数。

格式 [net，Ac，E]=adaptwb(NET，Pd，T，Ai，Q，TS)

info=adaptwb(code)

说明 adaptwb 函数是一个网络的自适应函数，该函数可以根据取学习函数来更新网络的权值和阈值。adaptwb 函数的参数为：

- NET 神经网络
- Pd 延迟输入
- T 每一层的目标向量
- Ai 初始输入条件
- Q 输入向量的个数
- TS 时间步长

网络的训练结束后，具有权值和阈值的学习函数的网络返回下列参数：

- net 更新后的网络
- Ac 总的层输出
- E 该层的误差

adaptwb(code)返回每一个 code 字符串代表的有用信息：

- 'pname' 训练参数的名称
- 'pdefaults' 默认的训练参数

参见 newp，newlin，train。

9. 训练函数

trainwb

功能 网络的权值和阈值的训练函数。

格式 [net，tr]=trainwb(NET，Pd，T，Ai，Q，TS，VV)

info=trainwb(code)

说明 trainwb 函数是一个网络训练函数，该函数根据其学习函数更新权值和阈值。trainwb 函数的输入参数为：

- NET 神经网络
- Pd 延迟输入（反馈）
- T 每一层的目标向量
- Ai 初始输入条件
- Q 输入向量的个数
- TS 时间步长
- VV 或者是空矩阵[ ]或者是确定的向量的结构

该函数返回的参数为：

- net 训练后的网络
- TR 每一步中各个值的训练记录：
  - TR.epoch 训练次数
  - TR.perf 训练性能
  - TR.vperf 验证性能
  - TR.tperf 测试性能

参见 train。

2.3 BP 网络的神经网络工具函数

MATLAB 提供了大量的 BP 网络工具函数，本节将对这些函数的功能、调用格式，以及使用方法做详细的介绍。

2.3.1 MATLAB 6.5 中有关 BP 网络的重要函数

MATLAB 6.5 的 BP 网络工具箱中包含了进行 BP 网络分析和设计的许多工具函数，表 2-2 列出了这些函数的名称和基本功能。

表 2-2 BP 网络的重要函数和功能

函数名	功能
deltalin	Purelin 神经元的 δ (delta)函数
deltalog	Logsig 神经元的 δ 函数
deltatan	Tansig 神经元的 δ 函数
errsurf	计算误差曲面
initff	最多三层的前向网络初始化
learnbp	反向传播学习规则

（续表）

函 数 名	功 能
Learnbpm	利用冲量规则的改进 BP 算法
learnlm	Levenberg-Marquardt 学习规则
logsig	对数 S 型传递函数
nwlog	对 Logsig 神经元产生 Nguyen-Midrow 随机数
nwtan	对 Togsig 神经元产生 Nguyen-Midrow 随机数
purelin	线性传递函数
simuff	前向网络仿真
tansig	正切 S 型传递函数
trainbp	利用 BP 算法训练前向网络
trainbpx	利用快速 BP 算法训练前向网络
trainlm	利用 Levenberg-Marquardt 规则训练前向网络

2.3.2 工具函数详解

工具函数主要包括神经元上的传递函数、（Delta）函数、基本函数、学习规则函数、绘图函数，以及误差分析函数等。

1. 神经元上的传递函数

（1）purelin

功能 线性传递函数。

格式 A = purelin(N)

A= purelin(z，b)

info = purelin(code)

说明 purelin 是一个从网络的输入计算出层的输出的线性传递函数，利用 Widrow-Holf 或 BP 算法训练的神经元的传递函数经常采用线性函数。其参数含义如下：

N N×Q 维的输入向量

神经元可采用的最简单的函数，即线性函数，它只是简单地将神经元输入经阈值调整后传递到输出。

purelin(N)函数可得到输入矢量为 N 时网络层输出矩阵

purelin(z，b)函数可用于成批处理矢量，并且提供阈值的情况。这时阈值矢量 b 与加权输入矩阵 z 是区分的。阈值矢量 b 加到 z 的每个矢量中去，以形成网络的输入矩阵，调用后得到的结果为矩阵。

在 purelin(p)函数中，p 指出传递函数特性的名称，调用后可得到所询问的特性，即：

- purelin('delta') 指出 delta 函数名称
- purelin('init') 指出标准初始化函数名称

- purelin('name')　指出传递函数的全称
- purelin('output')　指出包含传递函数最大和最小输出值的二元矢量

例程 2-1 的代码将产生一个线性传递函数并绘图表示。

例程 2-1

```
n = -5:0.1:5;
a = purelin(n);
plot(n,a)
```

输出结果如图 2-1 所示。

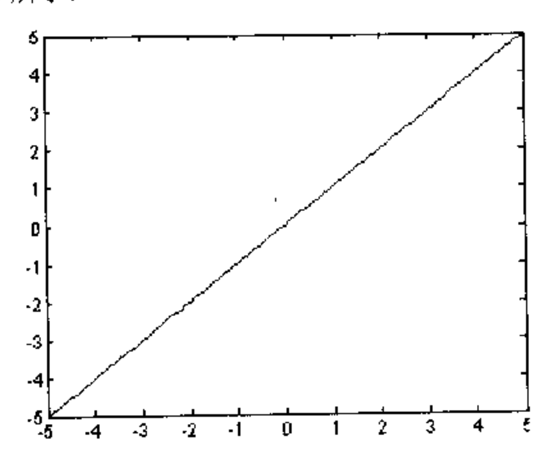


图 2-1　线性传递函数

参见　sim，dpurelin，satlin，satlins。

（2）tansig

功能　双曲正切 S 型（Sigmoid）传递函数。

格式　A = tansig(N)

A= tansig(z，b)

info = tansig(code)

说明　双曲正切 S 型（Sigmoid）传递函数如图 2-2 所示。

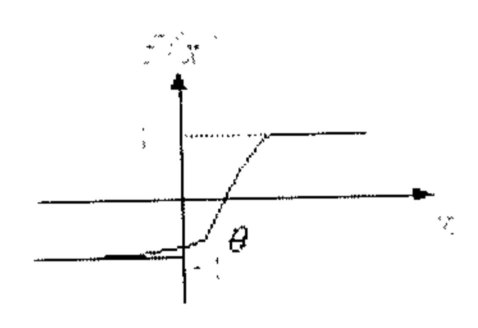


图 2-2　双曲正切 S 型（Sigmoid）传递函数

双曲正切 Sigmoid 传递函数用于将神经元的输入范围（$-\infty$，$+\infty$）映射到（-1，$+1$）。正切 Sigmoid 传递函数是可微函数，因此很适合于利用 BP 算法训练神经网络。该函数的各个参数的含义同 purelin。

例程 2-2 将产生一个 S 型传递函数并绘图表示。

例程 2-2

```
n = -5:0.1:5;
a = tansig(n);
plot(n,a)
```

输出如图 2-3 所示。

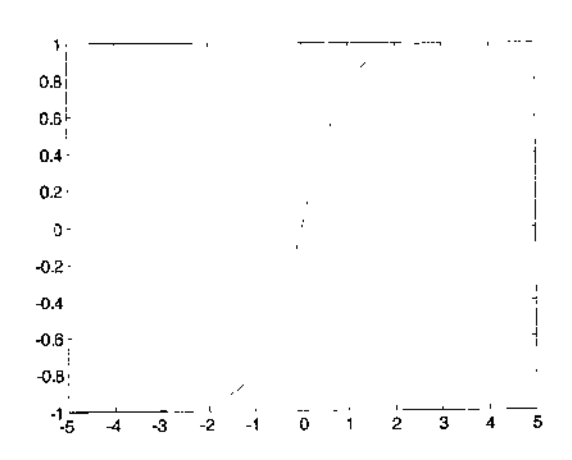


图 2-3　S 型传递函数

参见　sim，dtansig，logsig。

（3）logsig

功能　对数 S 型（Sigmoid）传递函数。

格式　A = logsig(N)

A= purelin(z，b)

info = logsig(code)

说明　Sigmoid 传递函数如图 2-4 所示。

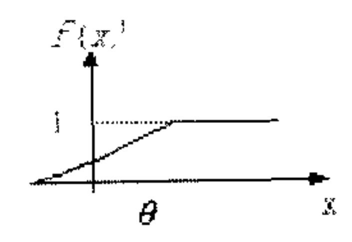


图 2-4　对数 Sigmoid 传递函数

对数 S 型（Sigmoid）传递函数用于将神经元的输入范围（－∞，＋∞）映射到（0，+1）。对数 S 型（Sigmoid）传递函数是可微函数，因此很适合于利用 BP 算法训练神经网络。该函数的各个参数的含义同 purelin。

例程 2-3 将产生一个对数 S 型传递函数并绘图表示。

例程 2-3

```
n = -5:0.1:5;
a = logsig(n);
plot(n,a)
```

输出结果如图 2-5 所示。

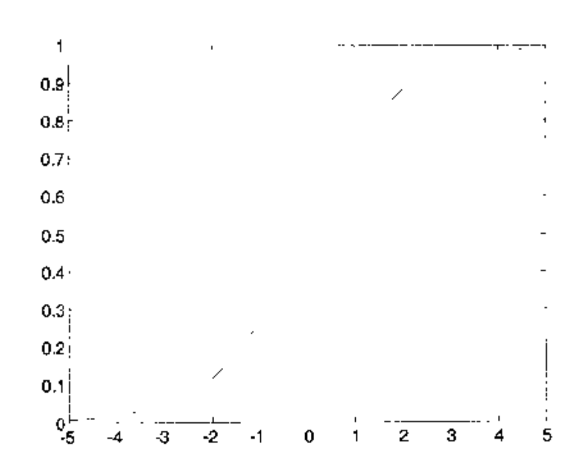


图 2-5 对数 S 型传递函数

参见 sim，dlogsig，tansig。

2．δ (Delta) 函数

（1）deltalin

功能 纯线性（purelin）神经元的 δ 函数。

格式 A=deltalin(a)

A=deltalin(a,e)

A=deltalin(a，d2，w2)

说明 通常的反向传播算法（BP）是利用网络误差平方和对网络各层输入的导数来调整其权值和阈值的，从而降低误差平方和。从网络误差矢量中可推导出输出层的误差导数或 δ（Delta）矢量，隐层的 δ 矢量导出，这种 δ 矢量的反向传播正是 BP 算法的由来。

deltalin(a)可计算出这一层输出对本层输出的导数，参数矩阵 a 为纯线性层的输出矢量。

deltalin(a,e) 可计算出线性输出层的误差导数，参数矩阵 a 和 e 分别为该层的输出矢量和误差。

deltalin(a，d2，w2)可计算出线性隐层的误差导数，参数矩阵 a 为纯线性层的输出矢量，d2 为下一层的 δ 矢量，w2 为与下一层的连接权值。

参见 无。

（2）deltalog

功能 对数 S 型（logsig）神经元的 δ 函数。

格式 A=deltalog(a)

A=deltalog(a,e)

A=deltalog(a，d2，w2)

说明 deltalog(a) 可计算出这一层输出对本层输出的导数，参数矩阵 a 为对数 S 型层的输出矢量。

deltalog(a,e) 可计算出 logsig 输出层的误差导数，参数矩阵 a 和 e 分别为该层的输出矢量和误差。

deltalog(a，d2，w2) 可计算出 logsig 隐层的误差导数，参数矩阵 a 为纯线性层的输出矢量，d2 为下一层的 δ 矢量，w2 为与下一层的连接权值。

参见 无。

（3）deltatan

功能 正切 S 型（tansig）神经元的 δ 函数。

格式 A=deltatan(a)

A=deltatan(a,e)

A=deltatan(a，d2，w2)

说明 deltatan(a) 可计算出这一层输出对本层输出的导数，参数矩阵 a 为正切 S 型层的输出矢量。

deltatan(a,e) 可计算出 tansig 输出层的误差导数，参数矩阵 a 和 e 分别为该层的输出矢量和误差。

deltatan(a，d2，w2) 可计算出 tansig 隐层的误差导数，参数矩阵 a 为正切 S 型层的输出矢量，d2 为下一层的 δ 矢量，w2 为与下一层的连接权值。

参见 无。

3．基本函数

（1）initff

功能 前向网络初始化。

格式 [w，b]=initff(p，S，f)

[w1，b1，w2，b2]=initff(p，S1，f1，S2，f2，S3，f3)

[w，b]=initff(p，S，t)

[w1，b1，w2，b2]=initff(p，S1，f1，S2，t2)

[w1，b1，w2，b2，w3，b3]=initff(p，S1，f1，S2，f2，S3，t3)

说明 initff(p，S，f)可得到 *S* 个神经元的单层神经网络的权值和阈值，其中 p 为输入矢量，f 为神经网络层间神经元的传递函数。



p 中的每一行中必须包含网络期望输入的最大值和最小值，这样才能合理地初始化权值和阈值。

initff 可对最多三层神经网络进行初始化，可得到每层的权值及阈值。

此外，initff 也可用目标矢量 t 代替网络输出层的神经元数，这时输出层的神经元数目就为 t 的行数。

参见　无。

（2）simuff

功能　前向网络仿真。

格式　simuff（p，w1，b1，f1）

simuff (p，w1，b1，f1，w2，b2，f2)

simuff(p，w1，b1，f1，w2，b2，f2，w3，b3，f3)

说明　前向网络由一系列网络层组成，每一层都从前一层得到输入数据，simuff 函数可仿真最多三层的前向网络。

参见　无。

（3）trainbp

功能　利用 BP 算法训练前向网络。

格式　[w，b，te，tr]=trainbp(w，b，'f'，p，t，tp)

[w1，b1，w2，b2，te，tr]=trainbp(w1，b1，'f1'，w2，b2，'f2'，p，t，tp)

[w1，b1，w2，b2，w3，b3，te，tr]=

trainbp(w1，b1，'f1'，w2，b2，'f2'，w3，b3，'f3'，p，t，tp)

说明　利用 BP 算法训练前向网络，使网络完成函数逼近、矢量分类及模式识别。

[w，b，te，tr]=trainbp(w，b，'f'，p，t，tp)利用单层神经元连接的权值矩阵 w，阈值矢量 b 及传递函数名 f 成批训练网络，使当输入矢量为 p 时，网络的输出为目标矢量矩阵 t。

tp 为可选训练参数，其作用是设定如何进行训练，具体如下：

- tp(1)　显示间隔次数，其默认值为 25
- tp(2)　最大循环次数，其默认值为 100
- tp(3)　目标误差，其默认值为 0.02
- tp(4)　学习速率，其默认值为 0.01

值得注意的是，当指定了 tp 参数时，任何默认或 NaN 值都会自动取其默认值。

当训练达到了最大的训练次数，或者网络误差平方和降到期望误差之下时，都会使网络停止学习。学习速率会影响权值与阈值更新的比例，较小的学习速率会导致学习时间增加，但有时可避免训练过程发散。

调用 trainbp 函数可得到新的权值矩阵 w、阈值矢量 b、网络训练的实际训练次数 te 及网络训练误差平方和的行矢量 tr。

对于多层网络，在调用 trainbp 函数时，可得到各层的权值矩阵及各层的阈值矢量。

参见　无。

（4）trainbpx

功能 利用快速 BP 算法训练前向网络。

格式 [w，b，te，tr]=trainbpx(w，b，'f'，p，t，tp)

[w1，b1，w2，b2，te，tr]=trainbpx(w1，b1，'f1'， w2，b2，'f2'，p，t，tp)

[w1，b1，w2，b2，w3，b3，te，tr]=

trainbpx(w1，b1，'f1'， w2，b2，'f2'， w3，b3，'f3'，p，t，tp)

说明 当采用动量时，BP 算法可找到更优的解；当采用自适应学习速率时，BP 算法可缩短训练时间，trainbpx 函数利用这两种方法来训练多层前向网络。

[w，b，te，tr]=trainbpx(w，b，'f'，p，t，tp) 利用单层神经元连接的权值矩阵 w，阈值矢量 b 及传递函数名 f 成批训练网络，使当输入矢量为 p 时，网络的输出为目标矢量矩阵 t。

tp 为可选训练参数，其作用是设定如何进行训练，具体如下：

- tp(1) 显示间隔次数，其默认值为 25
- tp(2) 最大循环次数，其默认值为 100
- tp(3) 目标误差，其默认值为 0.02
- tp(4) 学习速率，其默认值为 0.01
- tp(5) 设定学习速率增加的比率，其默认值为 1.05
- tp(6) 设定学习速率减少的比率，其默认值为 0.7
- tp(7) 设定动量常数，其默认值为 0.9
- tp(8) 设定最大误差比率，其默认值为 1.04



当指定了 tp 参数时，任何默认或 NaN 值都会自动取其默认值。

当训练达到了最大的训练次数，或者网络误差平方和降到期望误差之下时，都会使网络停止学习。自适应学习速率先给一个初始值，然后利用乘法使之增加或减少，以保持学习速度快而且稳定，定义在 0～1 之间的动量指定了所使用的动量大小，误差比率则限制了单次训练中可能增加的误差，如果误差上升超过了误差比率，则舍弃新的权值，并暂时不使用动量。

调用 trainbpx 函数可得到新的权值矩阵 ***w***、阈值矢量 ***b***、网络训练的实际训练次数 ***te*** 及矩阵 ***tr***，***tr*** 的第一行为训练过程中网络的误差，第二行为相应的自适应学习速率。当以二层或三层网络的权值矩阵、阈值矢量及传递函数调用 trainbpx 函数时，可得到各层的权值矩阵及各层的阈值矢量。

参见 无。

（5）trainelm

功能 训练 Elman 递归网络。

格式 [w1，b1，w2，b2，te，tr]=trainbpx(w1，b1，w2，b2，p，t，tp)。

说明 Elman 神经网络由一个正切 S 型隐层和一个纯线性输出层组成。Tansig 层接受网络输入及从自身的反馈信号，纯线性层从 Tansig 层得到输入。由于 Elman 神经网络为 S

型/线性网络，因此它能实现任何有限值函数。由于它有反馈连接，因此通过学习可识别或产生时间模式和空间模式。

在[w1，b1，w2，b2，te，tr]=trainbpx(w1，b1，w2，b2，p，t，tp)函数中，w1，b1 为 tansig 层的权值和阈值，w2，b2 位线性输出层的权值和阈值，p 为输入矢量，t 为相应的目标矢量，网络采用快速 BP 算法训练，以使产生相应于矢量 p 的输出矢量 t。

tp 为可选训练参数，其作用是设定如何进行训练，具体如下：

- tp(1) 显示间隔次数，其默认值为 5
- tp(2) 最大循环次数，其默认值为 500
- tp(3) 目标误差，其默认值为 0.01
- tp(4) 设置初始自适应学习速率，其默认值为 0.001
- tp(5) 设定学习速率增加的比率，其默认值为 1.05
- tp(6) 设定学习速率减少的比率，其默认值为 0.7
- tp(7) 设定动量常数，其默认值为 0.95
- tp(8) 设定误差比率，其默认值为 1.04



当指定了 tp 参数时，任何默认或 NaN 值都会自动取其默认值。

参见 无。

（6）trainlm

功能 利用 Levenberg-Marquardt 规则训练前向网络。

格式 [w，b，te，tr]=trainlm(w，b，'f'，p，t，tp)

[w1，b1，w2，b2，te，tr]=trainlm(w1，b1，'f1'，w2，b2，'f2'，p，t，tp)

[w1，b1，w2，b2，w3，b3，te，tr]=
trainlm(w1，b1，'f1'，w2，b2，'f2'，w3，b3，'f3'，p，t，tp)

说明 Levenberg-Marquardt 算法比 trainbp 和 trainbpx 函数使用的梯度下降法要快得多，但它需要更多的内存。

[w，b，te，tr]=trainlm(w，b，'f'，p，t，tp)利用训练参数 tp，输入矢量矩阵 p 和相应的目标矢量 t，对初始权值及阈值进行训练，从而得到新的权值 w 和阈值矢量 b。

tp 为可选训练参数，其作用是设定如何进行训练，具体如下：

- tp(1) 显示间隔次数，其默认值为 25
- tp(2) 最大循环次数，其默认值为 1000
- tp(3) 目标误差，其默认值为 0.02
- tp(4) 设置最小梯度，其默认值为 0.001
- tp(5) 设定 μ 的初始值，其默认值为 0.001
- tp(6) 设定 μ 的增加系数，其默认值为 10
- tp(7) 设定 μ 的减小系数，其默认值为 0.1
- tp(8) 设定 μ 的最大值，其默认值为 1e10



当指定了 tp 参数时，任何默认或 NaN 值都会自动取其默认值。

只有当训练中的误差达到了期望误差，μ 达到了最大值，或者达到了最大的训练次数时，才会停止训练。

变量 μ 确定了学习算法是根据牛顿法还是梯度法来完成的，下式为更新的 Levenberg-Marquardt 规则：

$$\Delta w = (J^T J + \mu I)^{-1} \cdot J^T e$$

随着 μ 的增大，Levenberg-Marquardt 中的项 $J^T J$ 可以忽略。因此学习过程主要根据梯度下降，即 $\mu^{-1} J^T e$ 项。只要迭代使误差增加，μ 也就会增加，直到误差不再增加为止。但是，如果 μ 太大，则会停止学习（因为 $\mu^{-1} J^T e$ 接近于 0），当已经找到最小误差时，就会出现这种情况，这就是为什么当 μ 达到最小值时要停止学习的原因。

参见　newff，newgd，traingdm，traingdm，traingda，traingdx。

4．学习规则函数

（1）learnbp

功能　反向传播学习规则函数。

格式　[dw，db]=learnbp(p，d，lr)。

说明　反向传播（BP）学习规则为调整网络的权值和阈值使网络误差的平方和最小，这是通过在最速下降方向上不断地调整网络的权值和阈值来达到的。

首先计算出网络输出层的误差矢量的导数，然后通过网络反向传播，直到计算出每个隐层的误差导数（称为 δ 矢量），接着可利用函数 deltalin，deltalog 和 deltatan 来计算。

根据 BP 准则，每一层的权值矩阵 $\boldsymbol{w}$ 利用本层的 $\boldsymbol{\delta}$ 矢量 $\boldsymbol{d}$ 和输入矢量 $\boldsymbol{p}$ 来更新：

$$\Delta w(i,j) = lr.\mathrm{d}(i).p(j)$$

其中 lr 为学习速率。

learnbp(p，d，lr)后，可得到权值修正矩阵，其中 $\boldsymbol{p}$ 为本层的输入矢量，$\boldsymbol{d}$ 为 δ 矢量，lr 为学习速率。

[dw，db]=learnbp(p，d，lr)可同时得到权值修正矩阵和阈值修正矢量。

参见　learngd。

（2）learnbpm

功能　利用动量规则的改进 BP 算法。

格式　[dw，db]=learnbpm(p，d，lr，mc，dw，db)。

说明　使权值的变化等于上次权值的变化与这次由 BP 准则引起的变化之和，这样可将动量加到 BP 学习中，上一次权值变化的影响可由动量常数来调整。当动量常数为 0 时，说明权值的变化仅由梯度决定。当动量常数为 1 时，说明新的权值的变化仅等于上次权值变化，而忽略掉梯度项，其数学表达式为：

$$\Delta w(i,j) = mc.\Delta w(i,j) + (1-mc).lr.d(i).p(j)$$

其中 mc 为动量常数。

调用 learnbpm(p，d，lr，mc，dw，db) 后，可得到权值修正矩阵，其中 p 为本层的输入矢量，d 为 δ 矢量，lr 为学习速率，mc 为动量常数，dw 为上一次权值变化矩阵。

[dw，db]=learnbpm(p，d，lr，mc，dw，db) 可同时得到新权值修正矩阵和新阈值修正矢量。

参见　learngdm。

（3）learnlm

功能　Levenberg-Marquardt 学习规则。

格式　learnlm(p，d)。

说明　这一函数用于计算网络输出误差对网络层权值的导数，利用 Levenberg-Marquardt 算法训练前向网络时（trainlm），要用到这种计算。Levenberg-Marquardt 算法的训练速度比梯度下降法要快得多，但需要更多的内存。

在 learnlm(p，d)函数中，p 为网络层的输入矢量，d 为每个网络误差对网络层每个输入的导数的雅可比矩阵（Jacobian），调用后可得到每个网络误差对网络层每个权值的导数的 Jacobian 矩阵。

参见　nnt2ff，train。

5．绘图函数

（1）plotes

功能　绘制误差曲面图。

格式　plotes(wv，bv，es)

plotes(wv，bv，es，v)

说明　plotes(wv，bv，es)绘制出由权值 wv 和阈值 bv 确定的误差曲面图 es（由 errsurf 产生），误差曲面图以三维曲面和等高线图形式显示。

plotes(wv，bv，es，v)允许设置期望的视角 v，默认值为[-37.5 30]。

例程 2-4 将绘制出误差曲面图，输出结果如图 2-6 所示。

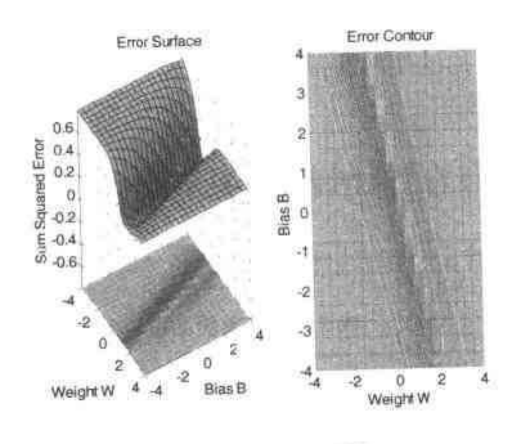


图 2-6　误差曲面图

例程 2-4

```
p = [3 2];
t = [0.4 0.8];
wv = -4:0.4:4; bv = wv;
ES = errsurf(p,t,wv,bv,'logsig');
plotes(wv,bv,ES,[60 30])
```

参见 errsurf。

（2）plotep

功能 在误差曲面上绘制出权值和阈值的位置。

格式 h = plotep(w,b,e)

h = plotep(w,b,e,h)

说明 plotep(w,b,e)在由 plotes 绘制的曲面图上，绘制出单输入神经元误差为 e 时的权值 w 和阈值 b 的位置。

当输出变量调用时，plotep 函数可返回包含有关位置信息的矢量 h。plotep(w,b,e,h)清除网络前一次位置，并重新画出新的神经元位置。

参见 errsurf，plotes。

6. 误差分析函数

errsurf

功能 计算误差曲面。

格式 e = errsurf(p,t,wv,bv,f)。

说明 errsurf(p,t,wv,bv,f)可计算单输入神经元误差曲面的平方和，输入为输入行矢量 p，相应的目标 t 和传递函数名 f。误差曲面是从每个权值与由权值 wv、阈值 bv 的行矢量确定的阈值的组合中计算出来的。

参见 plotes。

2.4 线性网络的神经网络工具函数

MATLAB 提供了大量的线性网络工具函数，本节将对这些函数的功能、调用格式，以及使用方法做详细的介绍。

2.4.1 MATLAB 6.5 中有关线性网络的工具函数

表 2-3 给出了 MATLAB 6.5 中与线性网络相关的神经网络工具函数。

表 2-3 线性网络的重要函数和功能

函数名称	功能
Newlind	设计一个线性层

（续表）

函数名称	功 能
Newlin	构造一个线性层
Purelin	线性传递函数
Dotprod	权值点积函数
Netsum	网络输入求和函数
initlay	某层的初始化函数
initwb	某层的权值和阈值的初始化函数
initzero	零权值阈值初始化函数
init	一个网络的初始化函数
mae	求平均绝对误差性能函数
learnwh	Widrow-hoff 的学习规则
adaptwb	网络的权值阈值的自适应函数
adaptwb	神经网络的自适应函数
trainwb	网络的权值和阈值训练函数
train	神经网络训练函数
maxlinlr	线性层的最大学习速率
errsurf	计算误差性能曲面
sim	仿真一个神经网络

2.4.2 工具函数详解

对于表 2-3 给出的函数中有些已经在 2.3 节中做过介绍，此处不再重述，下面分类介绍其余的函数。

1. 网络函数

（1）newlind

功能 设计一个线性层。

格式 net = newlind(P,T)。

说明 newlind(P,T)的输入参数含义为：

- P R×Q 维的 Q 组输入向量的矩阵
- T S×Q 维的 Q 组目标分类向量

该函数返回一个线性层，该线性层是将输入 P 设计为输出 T（具有最小均方误差和）。

例程 2-5 使用 newlind 函数来设计网络。

例程 2-5

```
%具有如下的给定输入P和输出目标T的线性层，并且检验其输出
P = [1 2 3];
T = [2.0 4.1 5.9];
net = newlind(P,T);
Y = sim(net,P)
```

输出：

```
Y =
     2.0500    4.0000    5.9500
```

newlind 函数通过求解下面的线性方程，根据输入 P 和目标 T 计算一个线性层的权值和阈值。

$$[W,b]*[P;ones]=T$$

参见 sim，newlin。

（2）newlin

功能 建立一个线性网络。

格式 net = newlin(PR,S,ID,LR)

new = newlin

说明 线性层常常在信号处理和预测中用做自适应滤波。newlin(PR,S,ID,LR)各个参数的含义为：

- PR　R 个输入元素的最大、最小值的矩阵（R×2）
- S　输出向量的个数
- ID　输入延迟向量，默认值=[0]
- LR　学习速率，默认值=0.01

该函数可返回一个新的线性层。

net = newlin(PR,S,0,P)函数用 0，P 取代了参数 ID，LR，其中 P 为输入向量的矩阵。此时函数返回一个线性层，该线性层具有对于输入 P 而言的最大的稳定学习速率。

参见 newlind，init，train。

2. 学习函数

learnwh

功能 Widrow-Hoff 学习函数。

格式 [dW,LS] = learnwh(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)

[db,LS] = learnwh(b,ones(1,Q),Z,N,A,T,E,gW,gA,D,LP,LS)

info = learnwh(code)

说明 learnwh 是 Widrow-Hoff 权值、阈值学习函数，也称为 delta 准则或最小方差准则。该函数可修改神经元的权值和阈值，使输出误差的平方和最小；可沿着误差平方和的最速下降方向连续调整网络的权值和阈值。由于线性网络的误差性能表面是抛物面，仅有一个最小值，故能保证网络是收敛的，只要学习速率不超出用 maxlinlr 函数计算的最大值。

learnwh(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)函数的参数含义为：

- W　S×R 权值矩阵（或 S×1 的阈值向量）
- P　R×Q 输入向量（或 Q 组单个输入）
- Z　S×Q 加权输入向量
- N　S×Q 网络的输入向量
- A　S×Q 输出向量
- T　S×Q 某层的目标向量
- E　S×Q 某层的误差向量
- gW　S×R 误差性能的梯度
- gA　S×Q 误差性能的输出梯度
- LP　学习参数，若无，LP=[ ]
- LS　学习状态，应初始化为[ ]

该函数返回：

- dW　S×R 权值（或阈值）变化矩阵
- LS　新的学习状态

网络按照 learnwh 的学习参数进行学习，下面给出默认值：

LP.lr=0.01　学习速率

Info=learnwh(code)函数针对每一种 code 代码返回相应的有用信息：

- 'pnames'　返回学习参数名
- 'pdefaults'　返回默认的学习参数
- 'needg'　若该函数使用 gW 或 gA，则返回 1

参见　newlin，adaptwb，trainwb，adapt

3. 分析函数

maxlinlr

功能　计算线性层的最大学习速率。

格式　lr = maxlinlr(P)

　　　lr = maxlinlr(P,'bias')

说明　maxlinlr 函数用于为 newlin 计算学习速率。通常学习速率越大，网络训练所需的时间越少。但是如果学习速率太大，学习过程就不稳定。该函数用来计算 Widrow-Hoff 算法的线性神经元层的最大学习速率。

maxlinlr(P)函数的参数为网络的给定输入 P 向量，可返回一个不带阈值的线性层所需的最大学习速率，该层仅用 P 中的向量进行训练。

maxlinlr(P,'bias')可返回一个带有阈值的线性层所需的最大学习速率。

下面定义了 4 组 2 维输入向量：

```
p=[1 2 -4 7;0.1 3 10 6]
```

对具有阈值的线性层求出其最大学习速率：

```
lr=maxlinlr(p,'bias')
```

上述语句执行后，结果如下：

lr=0.0067

参见 linnet，learnwh，adaptwh，trainwh。

2.5 自组织竞争网络工具箱函数

MATLAB 提供了大量的自组织竞争网络工具函数，本节将对这些函数的功能、调用格式，以及使用方法做详细的介绍。

2.5.1 MATLAB 6.5 中有关自组织网络的工具箱函数

MATLAB 6.5 中提供了许多进行神经网络设计和分析的工具箱函数，有关这些函数的使用可通过 help 命令得到。表 2-4 给出了与自组织网络有关的工具函数。

表 2-4 自组织网络的重要函数和功能

函数名称		功能
创建网络	newc	创建一个竞争层
	newsom	创建一个自组织特征映射
距离函数	Dist	欧氏距离权值函数
	mandist	Manhattan 距离权值函数
	linkdiat	Link 距离函数
初始化函数	Inttc	初始化竞争层
	initsm	初始化自组织特征映射网络
	init	初始化一个神经网络
	midpoint	中点权值初始化函数
学习函数	learnk	Kohonen 权值学习函数
	learnis	Instar 权值学习函数
	leatnos	Outstar 权值学习函数
	leatnsom	自组织特征映射权值学习函数
训练函数	trainc	训练竞争层
	trainsm	利用 Kohonen 规则训练自组织特征映射
	train	训练一个神经网络
仿真函数	simuc	竞争仿真层
	simusm	自组织特征映射网络仿真
	sim	仿真一个神经网络
领域函数	nbdist	用矢量距离表示的领域矩阵
	nbgrid	用栅格距离表示的领域矩阵
	nbman	用 Manhattan 距离表示的领域矩阵

（续表）

函数名称		功能
权值函数	negdist	对输入矢量进行加权计算
网络输入函数	netsum	计算网络输入矢量和
传递函数	compet	竞争传递函数
绘图函数	plotsm	绘制自组织特征映射网络的权值矢量
	plotsom	绘制自组织特征映射网络

2.5.2 工具函数详解

工具函数主要包括创建网络函数、距离函数、初始化函数、学习函数、训练函数、仿真函数、邻域函数、权值函数、传递函数，以及绘图函数等。

1. 创建网络

（1）newc

功能 用于创建一个竞争层。

格式 net = newc(P,,R,S,KLR)。

说明 newc 函数返回一个新的竞争层。其中，输入矢量矩阵 P 为 R×2 维矩阵，R 为输入向量的个数，并且在矩阵 P 中必须指明每一个输入矢量的最大、最小值范围。变量 S 表示神经元的个数；变量 KLR 表示 Kohonen 学习速率，其默认值为 0.01

参见 sim，init，adapt，train，trainwb1。

（2）newsom

功能 创建一个自组织特征映射。

格式 net = newsom(PR,[d1,d2,...,di]。

说明 newsom ( ) 函数返回一个新的自组织特征映射。其中，输入矢量矩阵 P 为 R×2 维矩阵，R 为输入向量的个数，并且在矩阵 P 中必须指明每一个输入矢量的最大、最小值范围；i 表示网络层的维数大小。

例程 2-6 创建一个自组织特征映射。

例程 2-6

```
%输入矢量分布在一个二维输入空间，其变化范围分别为[0 2]和[0 1]，组织特征映射网络维数为[3 5]，
%用 plotsom( )函数绘制组织特征映射。
P=[rand(1,400)*2;rand(1,400)];
net=newsom([0 2;0 1],[3 5]);
Plotsom(net.layers{1}.positions)
```

运行上面的代码，结果如图 2-7 所示。圆点表示神经元的位置，它们通过欧氏距离 1 相连。

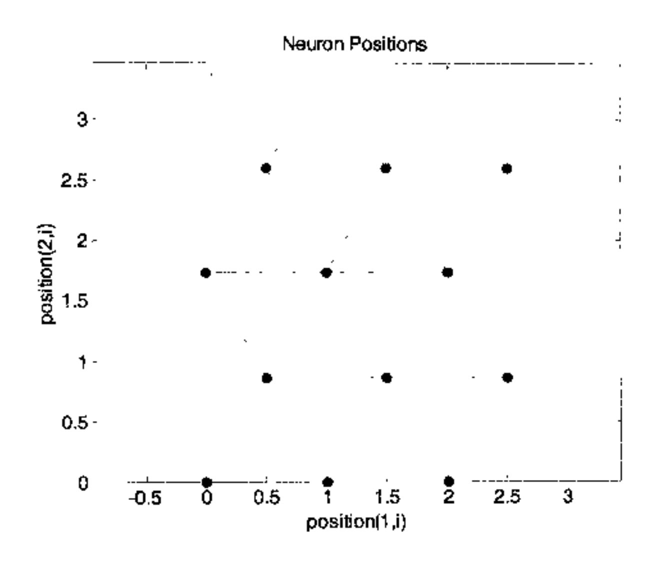


图 2-7 自组织特征映射

参见 sim，init，train。

2. 距离函数

（1）dist

功能 欧氏距离权值函数。

格式 Z = dist(W,P)

df = dist('deriv')

D = dist(pos)

说明 dist(W,P)函数是一个欧氏距离权值函数，它对输入进行加权，得到被加权的输入。其中，W 是权值函数，P 是输入矢量矩阵，函数返回一个矢量距离矩阵。一般而言，两个矢量 X 和 Y 之间的欧氏距离 d 定义为：

$$d = sum((x-y).\hat{}2).\hat{}5$$

D = dist(pos)函数也可以作为一个阶层距离函数，用于查找某一层神经网络中的所有神经元之间的欧氏距离，函数也返回一个距离矩阵。

任意给定输入矢量和权值矩阵，可利用 dist 函数计算欧氏距离。

```
W=rand(4,3);
P=rand(3,1);
Z=dist(W,P)
```

运行上面的代码，结果为：

```
Z =
0.5219
0.6374
0.2064
0.9015
```



参见 sim，dotprod，negdist，normprod，mandist，linkdist。

（2）mandist

功能 Manhattan 距离权值函数。

格式 Z = mandist(W,P)

df = mandist('deriv')

D = mandist(pos);

说明 mandist(W,P)函数是一个 Manhattan 距离权值函数，它对输入进行加权，得到被加权的输入。其中，W 是权值函数，P 是输入矢量矩阵，函数返回一个矢量距离矩阵。一般而言，两个矢量 X 和 Y 之间的 Manhattan 距离 d 定义为：

$$d = sum(abs(x - y))$$

D = mandist(pos)函数也可以作为一个阶层距离函数，用于查找某一层神经网络中的所有神经元之间的 Manhattan 距离距离，函数也返回一个距离矩阵。

任意给定输入矢量和权值矩阵，可利用 mandist 函数计算 Manhattan 距离。

```
W = rand(4,3);
P = rand(3,1);
Z = mandist(W,P)
```

运行上面的代码，结果为：

```
Z =
1.1677
0.4736
1.0733
1.1030
```

参见 dist，linkdist，sim。

（3）linkdist

功能 link 距离函数。

格式 d = linkdist(pos)。

说明 linkdist(pos)函数是一个阶层距离函数，用于查找某一层神经网络中的所有神经元之间的距离，函数返回一个距离矩阵。一般而言，对于 S 个矢量中，两个位置矢量 ***Pi*** 和 ***Pj*** 之间的 link 距离定义为：

$$Dij = \begin{cases} = 0, i = j \\ = 1, (sum((Pi - Pj).\verb|^|2)).\verb|^|.5 \leqslant 1 \\ = 2, kexit, Dik1 = Dk1k2 = Dk2j = 1 \\ = N, Dik1 = Dk1k2 = \ldots = Dknj = 1 \\ = S, others \end{cases}$$

例如，为分布在三维空间里的 10 个神经元任意定义一个位置矩阵，可以用 linkdist 函数查找其 link 距离。

```
pos = rand(3,10);
```

```
D = linkdist(pos)
```

执行上面的代码后，结果如下：

```
D =
0    1    1    1    1    1    1    1    1    1
1    0    1    1    1    1    1    1    1    1
1    1    0    1    2    1    1    1    1    1
1    1    1    0    2    1    1    1    1    1
1    1    2    2    0    1    1    2    1    2
1    1    1    1    1    0    1    1    1    1
1    1    1    1    1    1    0    1    1    1
1    1    1    1    2    1    1    0    1    1
1    1    1    1    1    1    1    1    0    1
1    1    1    1    2    1    1    1    1    0
```

参见 sim，dist，mandist。

3. 初始化函数

（1）initc

功能 初始化竞争层。

格式 w=initc(p，s)。

说明 initc(p，s)函数返回竞争层的权值。其中，p 为输入样本矢量矩阵，s 为神经元数。


输入矢量 p 的第 I 行需包含有网络输入 I 的最小值、最大值，这样才能正确初始化权值。

例程 2-7 利用 initc 函数初始化竞争层。

例程 2-7

```
p=[-3 3;0 6];
w=initc(p,3)
```

执行上面的代码后，结果如下：

```
w =
0    3
0    3
0    3
```

参见 无。

（2）initsm

功能 初始化自组织特征映射网络。

格式 w=initsm(p，s)。

说明 initsm(p，s)函数返回自组织特征映射网络的权值，其中，p 为输入样本矢量矩阵，s 为神经元数。必须注意，输入矢量 p 的第 I 行需包含有网络输入 I 的最小值、最大值，这样才能正确初始化权值。

例程 2-8 利用 initsm 函数初始化自组织特征映射网络。

例程 2-8

```
p=[-4 4;0 9];
w=initsm(p,3)
```

执行上面的代码后，结果如下：

```
w =
0      4.5000
0      4.5000
0      4.5000
```

参见 无。

（3）init

功能 初始化一个神经网络。

格式 net = init(net)。

说明 init(net)函数返回一个根据网络初始化函数更新了权值和偏差的神经网络。

参见 sim，adapt，initlay，initnw，initwb，rands。

（4）midpoint

功能 中点权值初始化函数。

格式 W = midpoint(S,PR)。

说明 在 midpoint(S,PR)函数中，S 为神经元的个数，P 为 R×2 维矩阵，它包含每个输入中的最大值和最小值，函数返回一个 S×2 维权值矩阵。它的每个值设定为（Pmin+Pmax）/2，因为网络输入很可能出现在中间区。因此，如果竞争层和自组织网络的初始权值选择在输入空间中的中间区，则其学习会更加有效。

例程 2-9 利用 midpoint 函数初始化权值。

例程 2-9

```
p=[0 1;-2 2];
w=midpoinit(3,p)
```

执行上面的代码后，结果如下：

```
w =
0.5000         0
0.5000         0
0.5000         0
```

参见 initwb，initlay，init。

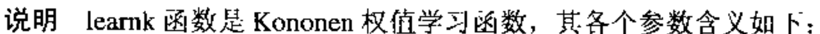
4. 学习函数

（1）learnk

功能 Kononen 权值学习函数。

格式 [dW,LS] = learnk(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)

info = learnk(code)



说明 learnk 函数是 Kononen 权值学习函数，其各个参数含义如下：

- W　S×R 权值矩阵（或 S×1 偏向量）
- P　R×Q 输入向量
- Z　S×Q 权值输入向量
- N　S×Q 网络输入向量
- A　S×Q 输出向量
- T　S×Q 层目标向量
- E　S×Q 层误差向量
- gW　S×R 性能梯度
- gA　S×Q 输出性能梯度
- D　S×S 神经元距离
- LP　学习参数，若无，则 LP=[ ]
- LS　学习状态，应初始化 LS=[ ]

该函数返回：

- dW　S×R 权变化矩阵
- LS　新的学习状态

Learnk()函数依据 Kononen 相关准则计算网络层的权变化矩阵。其学习通过调整神经元的权值等于当前输入矢量，使神经元存储输入矢量，用于以后的识别，即：

$$\Delta W(i,j) = lr * (P(j) - \Delta W(i,j))$$

参见 learnis,learnos。

（2）learnis

功能 Instar 权值学习函数。

格式 [dW,LS] = learnis(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)

　　info = learnis(code)

说明 learnis 函数依据 Instar 相关准则计算网络层的权变化矩阵。其学习用一个正比于神经元输出的学习速率来调整权值，学习一个新的矢量使之等于当前输入。这样，任何使 Instar 层引起高输出的变化，都会导致网络根据当前的输入矢量学习这种变化。最终，相同的输入使网络有明显不同的输出。即：

$$\Delta W(i,j) = lr * (P(j) - W(i,j)) * a(i)$$

learnis(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)函数各参数的含义如下：

- W　S×R 权值矩阵（或 S×1 偏向量）
- P　R×Q 输入向量
- Z　S×Q 权值输入向量
- N　S×Q 网络输入向量
- A　S×Q 输出向量
- T　S×Q 层目标向量

- E　S×Q 层误差向量
- gW　S×R 性能梯度
- gA　S×Q 输出性能梯度
- D　S×S 神经元距离
- LP　学习参数，若无，则 LP=[ ]
- LS　学习状态，应初始化 LS=[ ]

该函数返回：

- dW　S×R 权变化矩阵
- LS　新的学习状态

参见　learnk，learnos，adaptwb。

（3）learnos

功能　Outstar 权值学习函数。

格式　[dW,LS] = learnos(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)

info = learnos(code)

说明　learnos 函数依据 Outstar 相关准则计算网络层的权修改矩阵。Outstar 网络层的权可以看做是与网络层的输入矢量一样多的长期存储器。通常，Outstar 层是线性的，允许输入权值按线性层学习输入矢量。因此，存储在输入权值中的矢量可通过激活该输入而得到。即：

$$\Delta W(i,j) = lr*(a(i)-W(i,j))*P(j)$$

learnos 函数各个参数的含义同上。

例程 2-10 计算网络层的权变化矩阵。

例程 2-10

```
%一个自组织网络,在给定其随机输入矩阵 P,输出矩阵 A,权值矩阵 W 和学习速率后，可以用
earnos()
%函数计算其网络层的权变化矩阵。
p=rand(3,2);
a=rand(3,2);
w=rand(3,3);
lr=0.5;
dw=learnos(w,p,a,lr)
```

执行完上面的代码后，结果如下：

```
dw =
-0.3370    0.0232    0.0271
-0.3717   -0.2486   -0.3716
0.4561   -0.0667    0.5114
```

参见　learnis，learnk，adaptwb。

（4）learnsom

功能 自组织特征映射权值学习函数。

格式 [dW,LS] = learnk(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)

info = learnk(code)

说明 网络学习是根据 learnsom 函数所给出的学习参数开始的，其正常状态学习速率 LP.order_lr 默认值为 0.9，正常状态学习步数 LP.order_steps 默认值为 1000，调整状态学习速率 LP.tune_lr 默认值为 0.02，调整状态邻域距离 LP.tune_nd 默认值为 1。

learnos 函数各个参数的含义同上。

在网络处于正常状态和调整状态时，学习速率 lr 和邻域尺寸 nd 都得到更新。正常状态的持续时间由 LP.order_steps 定义，在这期间，学习速率 LR 从 LP.order_lr 调整到 LP.tune_lr，邻域尺寸从最大神经元距离下调到 1。神经元的权值按照期望的方向在适应神经元位置的输入空间建立次序。在调整状态，学习速率 LR 慢慢从 LP.tune_lr 减小，邻域距离 nd 常常被设为 LP.tune_nd。在这期间，神经元的权值按照期望的方向从输入空间伸展，直到保留到它们在正常状态时所建立的拓扑次序。

参见 learnis， adaptwb。

5. 训练函数

（1）trainsm

功能 利用 Kohonen 规则训练自组织特征映射网络。

格式 w=trainsm(w，b，p，tp) 。

说明 trainsm( )函数在训练参数 tp 的控制下，对输入矢量为 p 时的初始值进行训练，返回一个新的权值矩阵 w。其中，训练参数 tp 可设定为：

- tp(1) 表示两次更新显示的迭代次数，默认值为 25
- tp(2) 表示训练迭代的最大次数，默认值为 100
- tp(3) 表示初始学习速率，默认值为 1

例程 2-11 表示自组织特征映射网络训练。

例程 2-11

```
        %任意创建一个具有 100 个元素的输入矢量，构造一个排列在 3×3 栅格上由 9 个神经元组成
的自组。
        %织特征映射网络，对网络训练 400 次，观察网络的自组织能力。
        P=rand(2,100);
        W=initsm(P,9);
        M=nbman(3,3);
        W=trainsm(W,M,P,[20 400])
```

执行完上面的代码后，返回新的权值矩阵如图 2-8 所示。

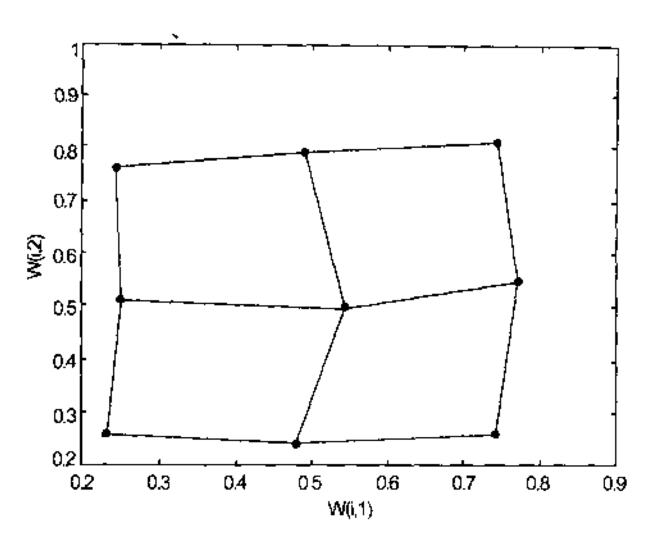


图 2-8　权值矩阵

```
W =
0.7460    0.8092
0.4927    0.7910
0.2446    0.7604
0.7751    0.5474
0.5460    0.4972
0.2532    0.5154
0.7440    0.2571
0.4827    0.2377
0.2341    0.2598
```

参见　无。

（2）trainc

功能　利用 Kohonen 规则训练自组织特征映射网络。

格式　w=trainc(w，b，　tp) 。

说明　trainc( )函数在训练参数 tp 的控制下，通过调整初始权值 w 和偏差矢量，返回相应于随机顺序选取的输入矢量 p 的新的权值矩阵。其中，训练参数 tp 可设定为：

- tp(1)　表示两次更新显示的迭代次数，默认值为 25
- tp(2)　表示训练迭代的最大次数，默认值为 100
- tp(3)　表示学习速率，默认值为 0.01
- tp(4)　表示跟踪性能（从 0 到 1），默认值为 0.999
- tp(5)　表示加权性能（从 0 到 1），默认值为 0.1

在竞争层的网络训练中，从一组输入矢量中随机选取一个矢量到竞争网络，直到满足指定的训练次数，然后找出输入最大的神经元，并利用 Kohonen 规则更新权值。在满足训练过程中，不断地调整偏差矢量，使得小概率输出为 1 的神经元具有较大的偏差，这可以促使那些“失败”的神经元有更多的调整机会。同时，训练参数 tp(4)越接近于 1，计算

神经元输出均值就越慢；训练参数 tp(5)越小，高或低的均值对神经元偏差和获胜率的影响也越小。因此，对于稳定的学习，tp(4)应非常接近于 1，tp(5)应非常小，然而如果 tp(4)太接近于 1，tp(5)太小，网络的训练时间也会增加。

参见 无。

（3）train

功能 利用 Kohonen 规则训练自组织特征映射网络。

格式 [net,tr] = train(NET,P,T,Pi,Ai)

[net,tr] = train(NET,P,T,Pi,Ai,VV,TV)

说明 train( )函数各个参数的含义如下：

- NET 神经网络
- T 网络目标矢量
- P 网络输入矢量
- Pi 初始网络延迟条件，默认值为 0
- Ai 初始层延迟条件，默认值为 0

返回值为：

- net 训练好的神经网络
- tr 有关训练的进程记录

参见 sim，init，adapt。

6. 仿真函数

（1）simuc

功能 竞争层仿真。

格式 simuc（p，w）。

说明 simuc（p，w）函数返回网络层的输出，其中 p 为输入矢量矩阵，w 为竞争层的权值矩阵。一个竞争层包含一层神经元，在任何给定时间，只有网络输入最大的神经元输出为 1，其他的神经元输出为 0。由于竞争层的任何一个输出向量中仅含惟一的非零值，因此 simuc（p，w）函数的返回值是一个稀疏矩阵，这就为计算机存储提供了方便，可以大大降低对机器内存的需求。

利用 simuc 函数可以计算出竞争层对输入的响应。

```
w=initc([0 2;-5 5],4);
a=simuc([2;4],w)
```

执行上面的代码后，其结果如下：

```
a =
(1,1)          1
```

参见 无。

（2）simusm

功能 自组织特征映射网络仿真。

格式 simusm（p，w，m）

simusm（p，w，m，n）

说明　simusm( )函数返回自组织特征映射网络的输出。其中 p 为输入矢量矩阵，w 为权值矩阵，m 为网络的邻阵，参数 n 为邻域的大小，默认值为 1。自组织特征映射网络由分布在一维或多维空间中的神经元组成，在任何时候，只有网络输入最大的神经元输出 1，与获胜神经元相邻的神经元输出为 0.5，其余神经元输出为 0。因此，对于网络输入最大的神经元 I，其对应的输出为 1，与之距离在 1 之内(m(i,j)≤ 1)的神经元 j，其输出为 0.5。

参见　无。

（3）sim

功能　网络层仿真。

格式　[T,X,Y]　　　= sim('model',timespan,options,ut)

　　　[T,X,Y1,...,Yn] = sim('model',timespan,options,ut)

说明　sim( )函数各个参数的含义如下：

- model　块的名称
- timespan　起始时间或[起始时间，终止时间]或[起始时间，输出时间，终止时间]
- options　最优仿真参数
- ut　最优外部输入
- T　返回的时间向量
- X　返回的状态矩阵或结构
- Y　返回的输出矩阵或结构

参见　sldebug，simset。

7. 邻域函数

（1）nbdist

功能　用矢量距离表示的邻域矩阵。

格式　nbdist（d1）

　　　nbdist(d1,d2)

　　　nbdist(d1,d2,…,d5)

说明　自组织网络中的神经元可以按照任何方式排列，这种排列可以用表示同一层神经元间距离的邻域来描述。因此，nbdist(d1)返回一维排列的 d1×d1 的邻域阵，表示一维中含有 d1 个神经元，其元素（i,j）表示神经元 i 与神经元 j 之间的矢量距离。nbdist(d1，d2)返回二维排列的（d1*d2）×(d1*d2)邻阵，表示二维中含有 d1*d2 个神经元，其元素（i,j）表示神经元 i 与神经元 j 之间的矢量距离。nbdist( )函数可用于每层最多有 5 维，最多有 5 层的网络，一般而言维数越少，收敛越快。运行命令：

m=nbdist（2，3）

程序运行结果为：

```
m =
0        1.0000   2.0000   1.0000   1.4142   2.2361
1.0000        0   1.0000   1.4142   1.0000   1.4142
2.0000   1.0000        0   2.2361   1.4142   1.0000
1.0000   1.4142   2.2361        0   1.0000   2.0000
1.4142   1.0000   1.4142   1.0000        0   1.0000
```

2.2361 1.4142 1.0000 2.0000 1.0000 0

参见 无。

（2）nbgrid

功能 用栅格距离表示的邻域矩阵。

格式 nbgrid(d1)

nbgrid(d1，d2)

nbgrid(d1，d2，...,d5)

说明 自组织网络中的神经元可以按照任何方式排列，这种排列可以用表示同一层神经元间距离的邻域来描述。而两神经元的栅格距离是指在神经元坐标相减后的矢量中，其元素幅值的最大值。因此，nbgrid(d1) 返回一维排列的 d1×d1 的邻域阵，表示一维中含有 d1 个神经元，其元素（i,j）表示神经元 i 与神经元 j 之间的栅格距离。nbdist(d1，d2) 返回二维排列的（d1*d2）×(d1*d2)邻阵，表示二维中含有 d1*d2 个神经元，其元素（i,j）表示神经元 i 与神经元 j 之间的栅格距离。nbgrid( )函数可用于每层最多有 5 维，最多有 5 层的网络，一般而言维数越少，收敛越快。

参见 无。

（3）nbman

功能 用 Manhattan 距离表示的邻域矩阵。

格式 nbman（d1）

nbman(d1，d2)

nbman(d1，d2，...,d5)

说明 自组织网络中的神经元可以按照任何方式排列，这种排列可以用表示同一层神经元间距离的邻域来描述。而两神经元的 Manhattan 距离是指在神经元坐标相减后的矢量中，其元素绝对值之和。因此，nbman(d1) 返回一维排列的 d1×d1 的邻域阵，表示一维中含有 d1 个神经元，其元素（i,j）表示神经元 i 与神经元 j 之间的栅格距离。nbman(d1，d2) 返回二维排列的（d1*d2）×(d1*d2)邻阵，表示二维中含有 d1*d2 个神经元，其元素（i,j）表示神经元 i 与神经元 j 之间的栅格距离。nbman( )函数可用于每层最多有 5 维，最多有 5 层的网络，一般而言维数越少，收敛越快。

参见 无。

8. 权值函数

negdist

功能 对输入矢量进行加权计算。

格式 Z = negdist(W,P)

df = negdist('deriv')

说明 negdist(W,P)函数中，W 表示 S×R 维权值矩阵，P 表示 R×Q 维输入矢量，函数返回 S×R 维负矢量距离矩阵。即：

$$Z = -sqrt(sum(w-p)^2)$$

例程 2-12 利用 negdist 函数计算加权输入。

例程 2-12

```
W = rand(4,3);
P = rand(3,1);
Z = negdist(W,P)
```

执行完上面的代码后，可得到如下结果：

```
Z =
-0.6637
-0.7414
-0.6095
-1.0426
```

参见　sim，dotprod，dist。

9. 传递函数

compet

功能　竞争传递函数。

格式　A = compet(n)

A = compet(z，b)

说明　compet(n)函数将神经元的网络输入进行转换，使网络输入最大的神经元输出为 1，而其余的神经元输出为 0。compet(n)函数返回一个输出矢量矩阵，其每一列矢量中仅包含一个 1，位于网络输入矢量 n 为最大的位置，而其余的元素为 0。compet(z，b)函数用于成批处理矢量、且偏差存在的情况下，偏差矢量 b 附加到加权输入矩阵 z 的每一个矢量上，形成网络输入矢量矩阵 n，然后利用竞争传递函数将输入矢量转换为输出矢量。

参见　sim，softmax。

10. 绘图函数

（1）plotsm

功能　绘制自组织特征映射网络的权值矢量。

格式　plotsm(W，M)。

说明　plotsm(W，M)函数给出了自组织网络的图形表示。在 W 中的每个神经元的权值行矢量相应的坐标处绘制出一点，然后表示相邻神经元权值的点之间根据邻阵 M 用线连接起来。即当 M(i,j)≤ 1 时，则将神经元 i 和 j 用线连接起来.

参见　无。

（2）plotsom

功能　绘制自组织特征映射网络。

格式　plotsom(pos)

plotsom(W，d，nd)

说明　其中参数 pos 表示 s 个 n 维神经元的位置。Plotsom(pos)函数绘图时，用红色的圆点表示神经元的位置，用欧氏距离 1 相连接。plotsom(W，d，nd)函数中，w 表示权值矩阵，d 表示距离矩阵，nd 表示邻域矩阵，其默认值为 1。它连接距离小于 1 的神经元的权值矢量。

参见 newsom，learnsom，initsom。

2.6 径向基网络神经网络工具箱函数

MATLAB 提供了大量的径向基网络工具函数，包括网络设计函数、权函数、网络输入函数、径向基传递函数、均方误差性能函数，以及变换函数等。本节将对这些函数的功能、调用格式，以及使用方法做详细的介绍。

1. 网络设计函数

（1）newrb

功能 设计一个径向基网络。

格式 net = newrb(P,T.GOAL,SPREAD)。

说明 用径向基函数网络逼近函数时，newrb 可自动增加径向基网络的隐层神经元，直到均方误差满足为止。

其中：

- P 输入矢量
- T 目标矢量
- GOAL 均方误差，默认时的值=0
- SPREAD 径向基函数的分布，默认时的值=1.0

参见 sim，newrbe，newgrnn，newpnn。

（2）newrbe

功能 设计严格的径向基网络。

格式 net = newrbe(P,T,SPREAD)。

说明 用径向基函数网络逼近函数时，newrbe 可迅速地设计一个径向基网络，且在设计中误差为 0。

其中：

- P 输入矢量
- T 目标矢量
- GOAL 均方误差，默认时的值=0
- SPREAD 径向基函数的分布，默认时的值=1.0

例程 2-13 设计一个径向基网络。

例程 2-13

```
%已知输入和目标矢量：
%      P = [1 2 3];
%      T = [2.0 4.1 5.9];
%设计一个径向基网络：
net = newrb(P,T);
%然后在网络的输入端输入一个新的值
P = 1.5;
```

```
%运用仿真函数 sim:
Y = sim(net,P)
```

输出结果：

```
Y =
2.6755
```

参见 sim，newrb，newgrnn，newpnn。

（3）newgrnn

功能 设计广义回归神经网络。

格式 net = newgrnn(P,T,SPREAD)。

说明 广义回归神经网络（GRNNs）是一种经常被应用在函数逼近中的径向基网络。newgrnn 函数可迅速设计 GRNNs 网络。

其中：

- P 输入矢量
- T 目标矢量
- SPREAD 径向基函数的分布，默认时的值=1.0

参见 sim，newrb，newpnn。

（4）newpnn

功能 设计概率神经网络。

格式 net = newpnn(P,T,SPREAD)。

说明 概率神经网络是一种适合于模式分类的径向基网络。

其中：

- P 输入矢量
- T 目标矢量
- SPREAD 径向基函数的分布，默认时的值=1.0

参见 ind2vec，vec2ind，newrb，newrbe，newgrnn。

2. 权函数

（1）dist

功能 Euclidean 距离权函数。

格式 Z = dist(W,P)

df = dist('deriv')

D = dist(pos)

说明 dist 是 Euclidean 距离权函数。所谓权函数是给定输入而会得到相应的权输入。两矢量 ***X*** 和 ***Y*** 之间的 Euclidean 距离 D 为：

D = sum((x-y).^2).^0.5

其中：

- W 为 S×R 权阵
- P 为 R×Q 的 Q 列输入矢量
- dist('deriv') 返回''，因为 dist 没有导数函数

- Pos 为神经元位置阵

例程 2-14 定义一随机矩阵 **W** 和输入矢量 **P**，计算相应的权 **Z**。

例程 2-14

```
W = rand(4,3);
P = rand(3,1);
Z = dist(W,P)
```

执行上面的代码后，结果如下：

```
Z =
0.6791
0.3994
0.3234
0.5746
```

参见 dotprod，negdist，normprod，mandist。

（2）dotprod

功能 点积权函数。

格式 Z = dotprod(W,P)

df = dotprod('deriv')

说明 dotprod 是点积权函数。所谓权函数是给定输入而会得到相应的权输入。

其中：

- W 为 S×R 权阵
- P 为 R×Q 的 Q 列输入矢量

参见 sim，ddotprod，dist，negdist，normprod。

（3）normprod

功能 规范点积权函数。

格式 Z = normprod(W,P)

df = normprod('deriv')

说明 normprod 是规范点积权函数。所谓权函数是给定输入而会得到相应的权输入。

其中：

- W 为 S×R 权阵
- P 为 R×Q 的 Q 列输入矢量

参见 dotprod，negdist，dist。

3. 网络输入函数

netprod

功能 网络输入的积函数。

格式 N = netprod(Z1,Z2,...)

df = netprod('deriv')

说明 netprod 是网络输入的积函数．网络输入函数结合它的加权输入和阈值计算一层的网络输出。

其中：

- Zi S×Q 维矩阵
- Netprod('deriv') 返回 netprod 的导数函数

参见 network/sim，dnetprod，netsum，concur。

4. 径向基传递函数

radbas

功能 径向基传递函数。

格式 A = radbas(N)

info = radbas(code)

说明 radbas 是径向基传递函数，次函数可由它的网络输入计算一层的输出。

其中：

- N 网络输入矢量矩阵
- Radbas('deriv') 函数可返回如下一些有用的信息：
  - 'deriv' 返回导数函数名称
  - 'name' 返回全名
  - 'output' 返回输出范围
  - 'active' 返回激活输入范围

例程 2-15 运用 plot 命令观察 radbas 传递函数的运行结果。

例程 2-15

```
n = -5:0.1:5;
a = radbas(n);
plot(n,a)
```

执行完上面的代码，其结果如图 2-9 所示。

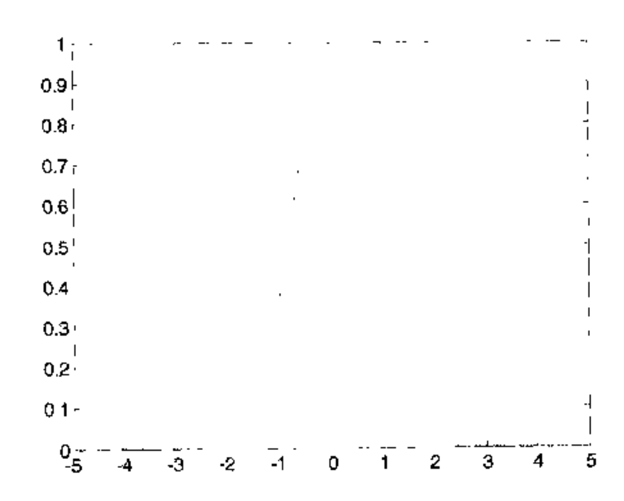


图 2-9 radbas 传递函数的运行结果

参见 sim，tribas，dradbas。

5. 均方误差性能函数

mse

功能 均方误差性能函数。

格式 perf = mse(e,x,pp)

perf = mse(e,net,pp)

info = mse(code)

说明 mse 是网络性能函数，此函数可按照均方误差测量网络性能。其中：

- e 为误差矢量
- x 为所有权值和阈值组成的矢量
- pp 为性能参数
- net 为所有权值和阈值能组成矢量 x 的神经网络
- mse(code)函数可返回如下一些有用的信息：
  - 'deriv' 为返回导数函数名称
  - 'name' 为返回全名
  - 'pnames' 为返回训练参数名
  - 'pdefaults' 为返回缺少的训练参数名

参见 msereg，mae，dmse。

6. 变换函数

（1）ind2vec

功能 将下标变换成单值矢量组。

格式 vec = ind2vec(ind)。

说明 ind2vec 函数输入为包含 n 个下标的行矢量 x(<=1)，调用后可得到 m 行 n 列的矢量组矩阵，结果是矩阵中的每个矢量 I，除了由 *x* 中的第 I 个元素指定的位置为 1 外，其余元素为 0，结果矩阵的行数 m 等于 x 中最大的下标值。

例如，定义一下标行矢量，利用 ind2vec 函数建起转换成单值矢量。

```
ind = [1 3 2 3];
vec = ind2vec(ind)
```

执行完上面的代码后，其结果如下：

```
vec =
(1,1)   1
(3,2)   1
(2,3)   1
(3,4)   1
```

参见 vec2ind。

（2）vec2ind

功能 将单值矢量组变换成下标矢量。

格式 ind = vec2ind(vec)。

说明 vec2ind 和 ind2vec 互为逆变化，vec2ind 函数输入为一个 m 行 n 列矢量矩阵 x，

调用后可得到 n 个下标值大小等于 0 的行矢量。X 中的每个矢量 I 除包含一个 1 外，其余均为 0，得到的行矢量包括这些非零元素的下标。

参见　ind2vec。

2.7　回归网络的神经网络工具箱函数

MATLAB 提供了两大类回归网络工具函数，即 Hopfield 神经网络和 Elman 神经网络，本节将对这些函数的功能、调用格式，以及使用方法做详细的介绍。

2.7.1　Hopfield 神经网络的工具箱函数

Hopfield 神经网络的工具箱函数包括网络函数和传递函数。

1. 网络函数

newhop

功能　设计一个 Hopfield 回归网络。

格式　net = newhop(T)。

说明　Hopfield 神经网络经常被应用于模式的联想记忆中。Hopfield 神经网络仅有一层，其输入用 netsum 函数，权函数用 dotprod 函数，传递函数用 satlins 函数。层中的神经元有来自它自身的连接权和阈值。其中 T 是目标矢量。

例程 2-16 设计一个 Hopfield 神经网络。

例程 2-16　设计一个 Hopfield 神经网络

```
%其目标矢量 T 为
T = [-1 -1 1; 1 -1 1]';
net = newhop(T);
%下面检验上述这个网络是否稳定在这些点上
Ai = T;
[Y,Pf,Af] = sim(net,2,[],Ai);
```

运行结果如下：

```
Y =
-1      1
-1     -1
1       1
Pf =
[ ]
Af =
-1      1
-1     -1
1       1
```

参见　sim，satlins。

2. 传递函数

satlins

功能 对称饱和线性传递函数。

格式 A = satlins(N)

info = satlins(code)

说明 对称饱和线性传递函数，传递函数由从网络的输入计算一层的输出。Satlins 函数的算法如下：

- 当 N≤－1 时，satlins(N)=－1。
- 当－1≤N≤1 时，satlins(N)=N。
- 当 1≤N 时，satlins(N)=1。

其中 N 为网络的输入矢量。

Satlins(code)返回如下信息：

- 'deriv' 返回导数函数名称
- 'name' 返回全名
- 'output' 返回输出范围
- 'active' 返回激活输入范围

例程 2-17 绘制 satlins 函数。

例程 2-17

```
n = -5:0.1:5;
a = satlins(n);
plot(n,a)
```

输出如图 2-10 所示。

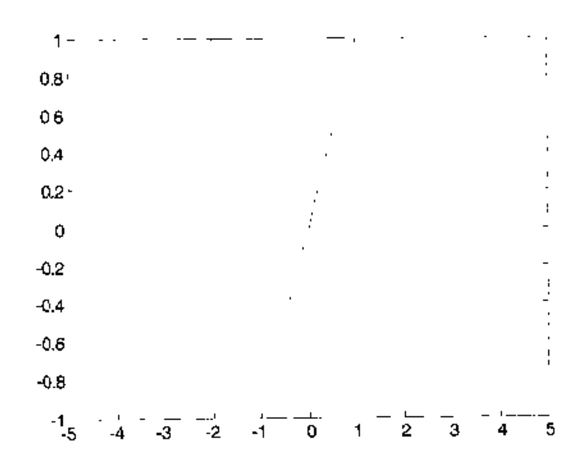


图 2-10 绘制 satlins 函数

参见 sim，satlin，poslin，purelin。



2.7.2　Elman 神经网络的工具箱函数

newelm

功能　生成一个 Elman 递归网络。

格式　net = newelm(PR,[S1 S2...SNl],{TF1 TF2...TFNl},BTF,BLF,PF)。

说明　Elman 神经网络由 N1 层组成，其权函数用 dotprod 函数，输入函数用 netsum 函数，以及用一些特殊的函数作为其传递函数。每层权和阈值初始化用 initnw。

其中：

- PR　为输入元素的最大和最小值的阵（其维数为：R×2）
- Si　为第 I 层的传递函数，默认时='tansig'
- BTF　为反向传播网络的训练次数(BTF 可以是 traingd，traingdm，traingda，traingdx 等函数)，默认时='traingdx'
- BLF　为反向传播权/阈值学习函数（BLF 可以是 learngd，learngdm），默认时='learngdm'
- PF　为性能分析函数（PF 可以是 mse 或 msereg），默认时='mse'

参见　newff，newcf，sim，init，adapt，train。

第 3 章　前向型神经网络理论及实例

前向型神经网络是整个神经网络体系中最常见的一种网络结构，其网络中各个神经元接受前一级的输入，并输出到下一级，网络中没有反馈，可以用一个有向无环路图表示。这种网络实现信号从输入空间到输出空间的变换，它的信息处理能力来自于简单非线性函数的多次复合。它的网络结构简单，易于实现。本章将对前向型神经网络的相关理论进行详细的介绍。

本章主要内容包括：

- 感知器网络
- BP 网络
- 线性神经网络
- 径向基函数网络
- GMDH 网络

3.1　感知器网络

感知器（Perceptron）是由美国学者罗森布拉特（F.Rosenblatt）于 1957 年提出的，它的目的是为了模拟人脑的感知和学习能力。感知器是最早提出的一种神经网络模型。

早期的研究人员试图用感知器模拟人脑的感知特征，但后来发现感知器的学习能力有很大的局限性（如只能对线性可分的输入向量进行分类），以至于人们对它的能力和应用前景得出了十分悲观的结论。尽管如此，这种神经网络模型的出现对早期神经网络的研究，以及后来许多神经网络的出现产生了极大的影响。就目前来看，它仍然是一种很有用的神经网络模型。感知器特别适用于简单的模式分类问题。

3.1.1　感知器模型

感知器是最基本的但具有学习功能的层状网络（Layed network）。最初的感知器有三层即 S（Sensory）层、A（Association）层和 R（Response）层组成，如图 3-1 所示。S 层和 A 层之间的耦合是固定的，只有 A 层和 R 层之间的耦合程度（即权值）可通过学习改变。本小节只讨论当 R 层（即输出层）只有一个节点的感知器时，它相当于单个神经元，简化结果如图 3-2 所示。但输入的加权和大于或等于阈值时，感知器的输出为 1，或者为 0 或－1，因此它可用于两类模式的分类。当两类模式可用一个超平面分开即线性可分时，权值 w 在学习中一定收敛，反之，则不收敛。Minsky 和 Papert 曾对感知器的分类能力做了严格的评价，并指出了它的局限性，例如它连最常用的异或（XOR）逻辑运算都无法实现。

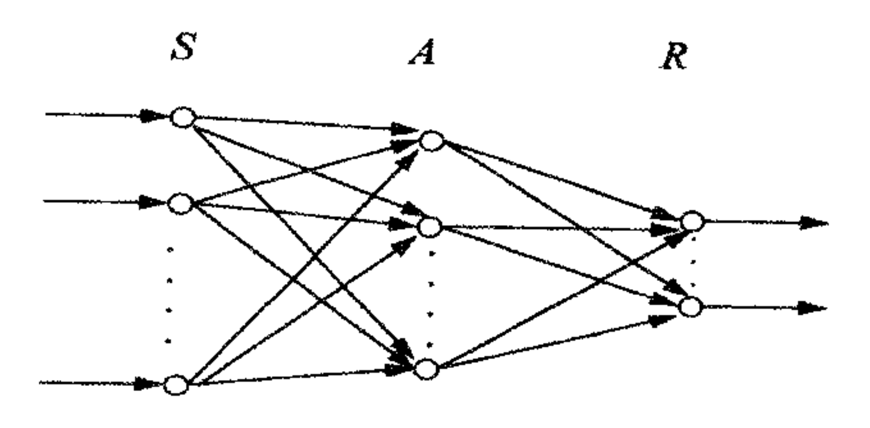


图 3-1　三层感知器

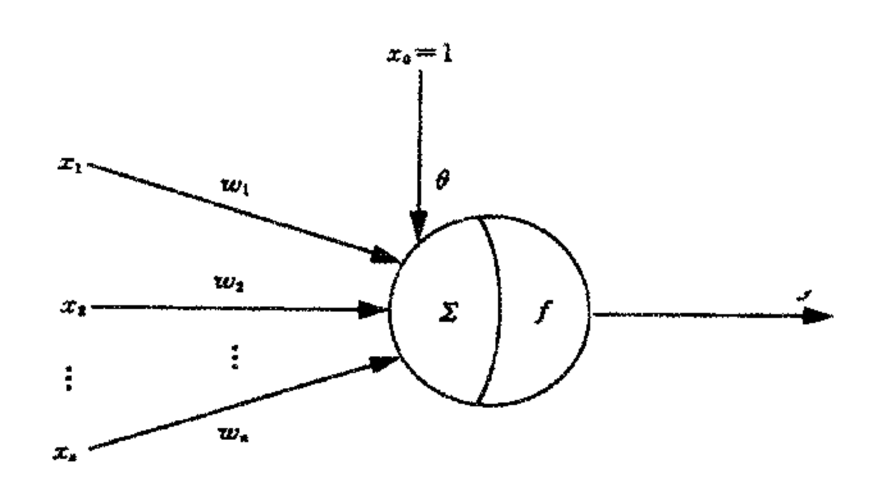


图 3-2　简化感知器

【定理 3.1】假定隐层的节点可以根据需要任意设置，那么用三层（不包括 S 层）的阈值网络可以实现任意的二值逻辑函数。

值得注意的是，感知器学习方法在函数不是线性可分时，得不出任何结果，另外也不能推广到一般的前向网络中去。其主要原因是传递函数为阈值函数，为此人们用可微函数如 Sigmoid 曲线来代替阈值函数，然后采用梯度算法来修正权值。BP 网络就是这样算法的典型网络。

MATLAB 6.5 提供的一种线性阈值单元为 hardlim。

例如，建立一个 hardlim 传递函数并且打印出该图形，如图 3-3 所示。

```
n=-5:0.1:5;
a=hardlim(n);
plot(n,a)
```

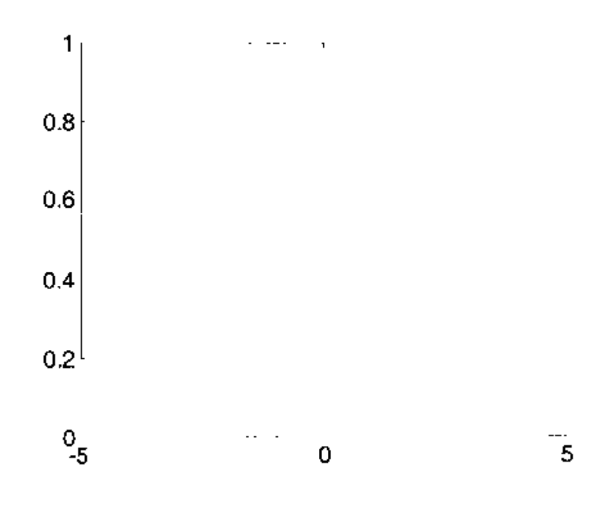


图 3-3 hardlim 传递函数

3.1.2 感知器神经元网络的网络结构

图 3-4 描述了感知器神经元网络结构的两种形式。感知器神经元网络是由 hardlim 产生的符号函数阈值元件组成的，该网络是由单层的 S 个神经元构成的。网络有 R 个输入，通过一组权值 $w(1,i)$ 与 S 个神经元组成，网络的输出为 S 个。

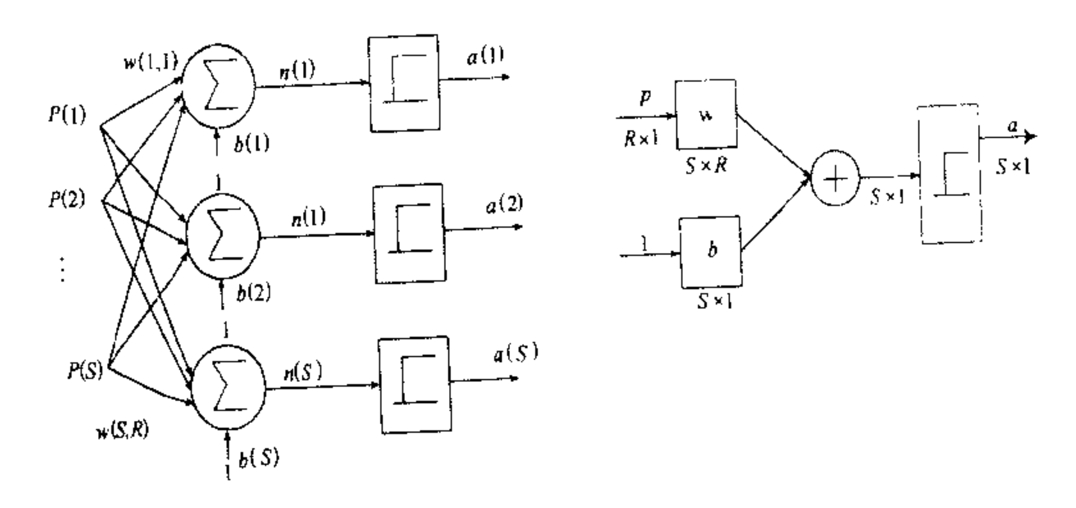


图 3-4 感知器神经元结构

从图 3-4 可以看出，感知器神经网络只有一层神经元，这是因为感知器的学习规则只能训练单层网络，根据网络结构，可以写出第 i 个输出神经元（ $i=1,\dots,S$ ）的输出：

$$n(i)=\sum_{j=1}^{R} w(i,j)\times p(j)+b(i)$$

$$a(i)=f[n(i)]$$

感知器中使用了符号阈值函数单元，因此使感知器能够将输入向量分为两个区域，则有：

$$a(i)=\begin{cases}1, & n(i)\geqslant 0\\ 0, & n(i)<0\end{cases}$$

当输入 $n(i)\geqslant 0$ 时，感知器输出 $a(i)$ 为 1，否则 $a(i)$ 输出为 0。

3.1.3 感知器神经网络的初始化

进行程序设计的第一步就是初始化，对感知器神经网络的初始化可采用 MATLAB 6.5 神经网络工具箱中的 init( )函数。使用 init(net)函数可以得到一个已初始化的神经网络，该网络的权值和阈值是按照网络初始化函数来进行调整的，而网络的初始化函数是由 NET.initFcn 设定的，其参数是由 Net.initParam 指定的。初始化函数的用法为：

net=init(net)

MATLAB 6.5 工具函数中的 newp 函数可以生成一个感知器网络，其中使用了上述语句对网络 net 进行初始化。

3.1.4 感知器神经网络的学习规则

学习规则是用来计算网络的新的权值和阈值的算法，感知器利用其学习规则来调整网络的权值和阈值，使该网络的输出最终达到目标的期望值。

对于输入向量 $\boldsymbol{P}$，输出向量 $\boldsymbol{a}$，目标矢量为 $\boldsymbol{t}$ 的感知器，该感知器的学习误差为 e，则 $e=t-a$。此时感知器的权值与阈值修正公式为：

$$\Delta w(i,j)=[t(i)-a(i)]*p(j)=e(i)*p(j)$$
$$\Delta b(i)=[t(i)-a(i)]*1=e(i)*1$$

式中 $i=1,2,\ldots,S$，$j=1,2,\ldots,R$，则更新的权值与阈值为：

$$w(i,j)=w(i,j)+\Delta w(i,j)$$
$$b(i)=b(i)+\Delta b(i)$$

用矩阵来表示相应的权值与阈值更新公式：

$$w=w+e*p^T$$
$$b=b+e$$

感知器的学习规则属于梯度下降法，该法则已被证明：如果能存在，这算法在有相持的循环迭代后可以收敛到正确的目标矢量。

在 MATLAB 6.5 的神经网络工具箱中，感知器的学习规则可用函数 learnp( )和 learnpn( )实现。其使用格式分别为：

```
[dW,LS] = learnp(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)
[dW,LS] = learnpn(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)
```

例如，在 2 个输入、3 个神经元的网络中，定义一个随机输入 p 和误差 e，来计算网络权值的改变：

```
p = rand(2,1);
e = rand(3,1);
dW = learnpn([],p,[],[],[],[],e,[],[],[],[],[])
```

结果如下：

```
dW = 0.4122    0.1003
0.3301    0.0803
0.6055    0.1473
```

3.1.5　感知器神经网络的训练

神经网络在应用之前必须经过训练，由此决定网络的权值和阈值。感知器的训练过程为：

对于给定的输入向量 $\boldsymbol{P}$，计算网络的实际输出为 a，并于相应的目标向量 t 进行比较，用得到的误差 e，根据学习规则进行权值和阈值的调整；重新计算网络在新权值作用下的输入，重复权值调整过程，直到网络的输出与目标值相等或者训练的次数达到预想设定的最大值时结束训练。

经过训练的网络需要验证其是否能够达到要求（即网络是否训练成功）。若网络训练成功，那么训练后的网络对于被训练的每一组输入向量都能产生一组对应的期望输出；若在给定的最大训练次数内，网络未能完成在给定输入向量 $\boldsymbol{P}$ 的作用下，满足 $a=t$ 的要求，那么可以增加训练的次数，继续训练网络；若在足够多次的训练后，网络仍然达不到要求，那么需要分析一下，感知器神经网络是否适合解决目前这个问题。感知器并非适用于任何分类问题。

在 MATLAB 6.5 的神经网络工具箱中，感知器的训练函数为 adaptwb( )和 trainwb()，其使用格式分别为：

```
[net,Ac,El] = adaptwb(net,Pd,T,Ai,Q,TS)
[net,tr] = trainwb(net,Pd,Tl,Ai,Q,TS,VV)
```

3.1.6　感知器的局限性

由于感知器神经网络在结构和学习规则上的局限性，其应用被限制在一定的范围。一般来说，感知器有以下局限性：

- 由于感知器的激活函数是阈值函数，则感知器神经网络的输出只能取 0 或 1。因此感知器只能适应于简单的分类问题。
- 感知器神经网络只能对线性可分的向量集合进行分类。理论上已经证明，只要输入向量是线性可分的，感知器在有限的时间内总能达到目标向量。但是如何确定

输入向量是否线性可分，尤其当输入向量增多时，更难以确定。一般只有设置一定的循环次数，对网络进行训练而判定它是否能被线性可分。

- 当感知器神经网络的所有输入样本中存在奇异的样本，即该样本向量同其他所有的样本向量比较起来特别大或特别小时，网络训练花费的时间将很长。

例如，输入和目标向量分别为：

p=[-0.5 -0.5 0.3 -0.1 -80;-0.5 0.5 -0.5 1.0 100]; t=[1 1 0 0 1];

由于输入的第五组数远远大于其他输入数据，这必然导致训练的困难。解决此问题的方法是采用标准化感知器学习规则。在神经网络工具箱中，实现标准化感知器学习规则的函数是 learnpn( )。

3.1.7 多层感知器

为了解决单层感知器的局限性，20 世纪 60 年代末人们致力于该问题的研究。并找到了解决的方法——采用多层网络结构。这样，对于单层感知器解决不了的“异或”问题，可以用两层网络结构得以解决。如图 3-5 所示为一种常用的双层感知器神经网络。其第一层是随机感知层，第二层为学习感知层。

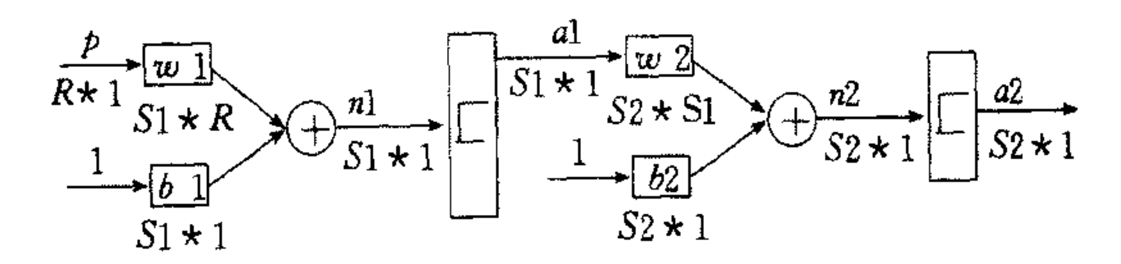


图 3-5 双层感知器网络结构图

3.2 BP 网络

学者 Rumelhart 和 McClelland 与他们的同事认识到了神经元网络在信息处理方面的重要性，于 1982 年成立了 PDP 小组，研究并行分布式式信息处理方法，探索人类认知的微结构。1986 年 Rumelhart，Hinton 和 Williams 完整而简明地提出一种 ANN 的误差反向传播训练算法（简称 BP 算法），系统地解决了多层网络中隐含单元连接权的学习问题，还对其能力和潜力进行了探讨。

目前，在人工神经网络的实际应用中，绝大部分的神经网络模型是采用 BP 网络和它的变化形式，它也是前向网络的核心部分，并体现了人工神经网络最精华的部分。

BP 网络主要应用于：

- 函数逼近　用输入矢量和相应的输出矢量训练一个网络逼近一个函数。
- 模式识别　用一个特定的输出矢量将它与输入矢量联系起来。

- 分类 把输入矢量以所定义的合适方式进行分类。
- 数据压缩 减少输出矢量维数以便于传输或存储。

3.2.1 BP 网络结构

BP 网络（Backpropagation NN）是一种单向传播的多层前向网络，其结构如图 3-6 所示。网络除输入输出节点外，还有一层或多层的隐层节点，同层节点中没有任何耦合。输入信号从输入层节点依次传过各隐层节点，然后传到输出节点，每一层节点的输出只影响下一层节点的输出。其节点单元特性（传递函数）通常为 Sigmoid 型(f(x)=1/(1+exp(-Bx))(B>0)，但在输出层中，节点的单元特性有时为线性。

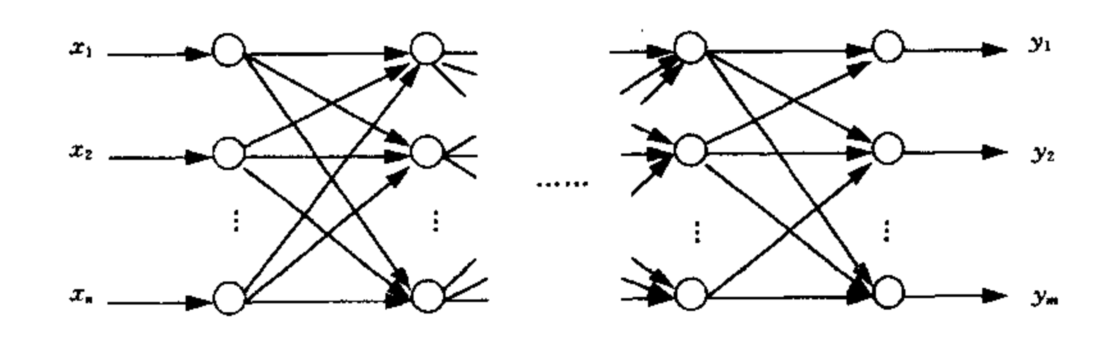


图 3-6 BP 网络

BP 网络可看做是一个从输入到输出的高度非线性映射，即 $F: R^n \to R^m$，$f(X)=Y$。对于样本集合：输入 $x_i(\in R^n)$ 和输出 $y_i(\in R^m)$，可认为存在某一映射 g 使：

$$g(x_i) = y_i \qquad i=1,2,\ldots,n$$

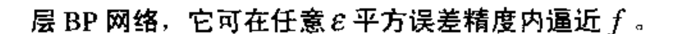
现要求求出一映射 f，使得在某种意义下（通常是最小二乘意义下），f 是 g 的最佳逼近。神经网络通过对简单的非线性函数进行数次复合，可近似复杂的函数。

下面介绍 Kolmogorov 定理，即映射网络存在定理。

【定理 3.2：Kolmogorov 定理】给定任一连续函数 $f:U^n \to R^m, f(X)=Y$，这里 U 是闭单位区间[0,1]，f 可以精确地用一个三层前向网络实现，次网络的第一层（即输入层）有 n 个处理单元，中间层有 $2n+1$ 个处理单元，第三层（即输出层）有 m 个处理单元。

尽管 Kolmogorov 定理保证任一连续函数可由一个三层前向网络来实现，但它没有提供任何构造这样一个网络的可行方法。下面给出 BP 定理，在 BP 网络中它可在任意希望的精度上实现任意的连续函数。

【定理 3.3：BP 定理】给定任意 $\varepsilon > 0$ 和任意 L_2 函数 $f:[0,1]^n \to R^m$，存在一个三



层 BP 网络，它可在任意 ε 平方误差精度内逼近 f。

虽然 BP 定理告诉我们，只要用三层的 BP 网络就可实现 L_2 函数，但实际上，还是有必要使用更多层的 BP 网络，其原因是用三层 BP 网络来实现 L_2 函数，往往需要大量的隐层节点，而使用多层网络可减少隐层节点数。但如何选取网络的隐层数和节点数，还没有确切的方法和理论，通常是凭对学习样本和测试样本的误差交叉评价的试错法选取。

3.2.2 BP 算法的数学描述

基于 BP 算法的多层前馈型网络的结构见图 3-7。

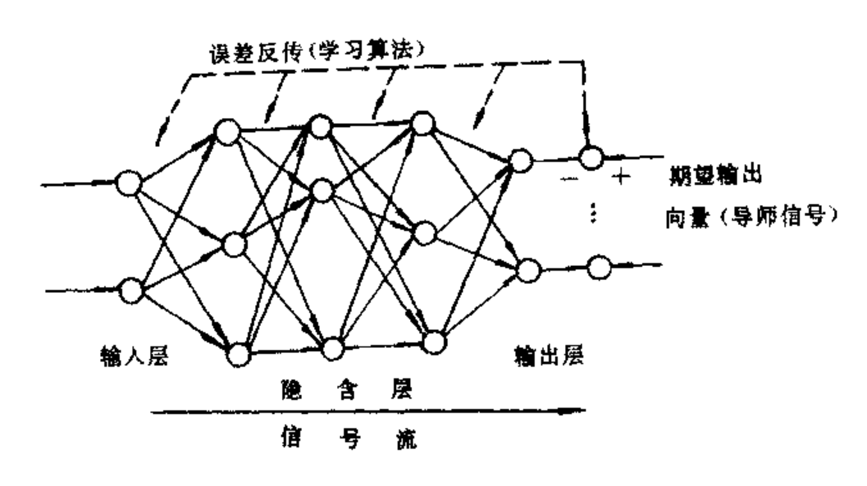


图 3-7 基于 BP 算法的神经元网络的结构

这种网络不仅有输入节点、输出节点，而且还有一层或多层隐含节点。对于输入信息，要先向前传播到隐含层的节点上，经过各单元的特性为 Sigmoid 型的激活函数（又称作用函数、转换函数或映射函数等）运算后，把隐含节点的输出信息传播到输出节点，最后给出输出结果。网络的学习过程由正向和反向传播两部分组成。在正向传播过程中，每一层神经元的状态只影响到下一层神经元网络。如果输出层不能得到期望输出，就是实际输出值与期望输出值之间有误差，那么转入反向传播过程，将误差信号沿原来的连接通路返回，通过修改各层神经元的权值，逐次地向输入层传播去进行计算，再经过正向传播过程，这两个过程的反复运用，使得误差信号最小。实际上，误差达到人们所希望的要求时，网络的学习过程就结束。

BP 算法是在导师指导下，适合于多层神经元网络的一种学习，它是建立在梯度下降法的基础上的。

设含有共 L 层和 n 个节点的一个任意网络，每层单元只接受前一层的输出信息并输出给下一层各单元，各节点（有时称单元）的特性为 Sigmoid 型（它是连续可微的，不同于感知器中的线性阈值函数，因为它是不连续的）。为简单起见，认为网络只有一个输出 y。设给定 N 个样本（x_k, y_k）（$k=1,2,\dots,N$），任一个节点 i 的输出为 O_i，对某一个输入为 x_k，网络的输出 y_k，节点 i 的输出为 O_{ik}，现在研究第 l 层的第 j 个单元，当输入第 k

个样本时，节点 j 的输入为：

$net_{jk}^{l}=\sum_{j} w_{ij}^{l} O_{jk}^{l-1}$，$O_{jk}^{l-1}$ 表示 $l-1$ 层，输入第 k 个样本时，第 j 个单元节点的输出：

$$O_{jk}^{l}=f(net_{jk}^{l})$$

使用误差函数为平方型：

$$E_k=\frac{1}{2}\sum_{l}(y_{jk}-\bar{y}_{jk})^2$$

$\bar{y}_{jk}$ 是单元 j 的实际输出。总误差为：

$$E=\frac{1}{2N}\sum_{k=1}^{N}E_k$$

定义
$$\delta_{jk}^{l}=\frac{\partial E_k}{\partial net_{jk}^{l}}$$

于是
$$\frac{\partial E_k}{\partial w_{ij}^{l}}=\frac{\partial E_k}{\partial net_{jk}^{l}}\frac{\partial net_{jk}^{l}}{\partial w_{ij}^{l}}=\frac{\partial E_k}{\partial net_{jk}^{l}}O_{jk}^{l-1}$$

下面分两种情况来讨论：

- 若节点 j 为输出单元，则 $O_{jk}^{l}=\bar{y}_{jk}$

$$\delta_{jk}^{l}=\frac{\partial E_k}{\partial net_{jk}^{l}}=\frac{\partial E_k}{\partial \bar{y}_{jk}}\frac{\partial \bar{y}_{jk}}{\partial net_{jk}^{l}}=-(y_k-\bar{y}_k)f'(net_{jk}^{l})$$

- 若节点 j 不为输出单元，则

$$\delta_{jk}^{l}=\frac{\partial E_k}{\partial net_{jk}^{l}}=\frac{\partial E_k}{\partial O_{jk}^{l}}\frac{\partial O_{jk}^{l}}{\partial net_{jk}^{l}}=\frac{\partial E_k}{\partial O_{jk}^{l}}f'(net_{jk}^{l})$$

式中 O_{jk}^{l} 是送到下一层 $(l+1)$ 的输入，计算 $\frac{\partial E_k}{\partial O_{jk}^{l}}$ 要从 $(l+1)$ 层算回来。

在 $(l+1)$ 层第 m 个单元时：

$$\therefore\quad \frac{\partial E_k}{\partial O_{jk}^{l}}=\sum_{m}\frac{\partial E_k}{\partial net_{jk}^{l}}\frac{\partial net_{jk}^{l}}{\partial O_{jk}^{l}}=\sum_{m}\frac{\partial E_k}{\partial net_{mk}^{l+1}}w_{mj}^{l+1}=\sum_{m}\delta_{mk}^{l+1}w_{mj}^{l+1}$$

由以上两式可以得到：

$$\delta_{jk}^{l}=\sum_{m}\delta_{mk}^{l+1}w_{mj}^{l+1}f'(net_{jk}^{l})$$

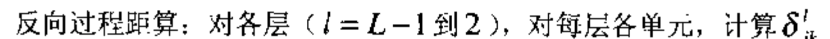
现在，反向传播算法的步骤可概括如下：

（1）选定权系数初值；

（2）重复下述过程直到收敛：

① 对 $k=1$ 到 N

正向过程计算：计算每层各单元的 O_{jk}^{l-1}，net_{jk}^{l} 和 $\bar{y}_k$，$k=2,\dots,N$



反向过程距算：对各层（$l=L-1$ 到 2），对每层各单元，计算 δ_{jk}^{l}

② 修正权值

$$w_{ij}=w_{ij}-\mu\frac{\partial E}{\partial w_{ij}} \quad \mu>0$$

μ 为步长，其中 $$\frac{\partial E}{\partial w_{ij}}=\sum_{k=1}^{N}\frac{\partial E_k}{\partial w_{ij}}$$

3.2.3 BP 网络中的神经元模型

图 3-8 给出一个基本的神经元模型，它具有 R 个输入，每个输入都通过一个适当的权值 w 与神经元相连，神经元的输出可表示成：

$$a=f(w*p,b)$$

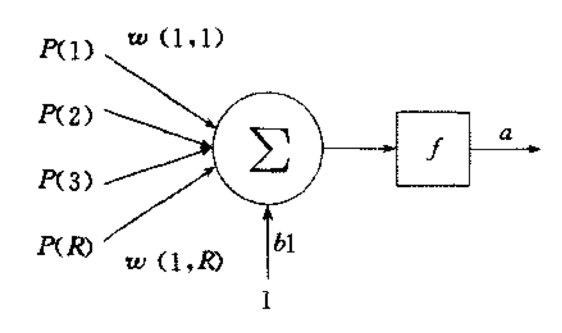


图 3-8 基本的神经元模型

BP 网络中基本神经元的激活函数必须处处可微，所以，经常使用的是 S 型的对数或正切激活函数或线性函数。

3.2.4 BP 网络的训练过程

为了应用神经网络，在选定所要设计的神经网络的结构之后（其中包括的内容有网络的层数、每层所含神经元的个数和神经元的激活函数），首先应考虑神经网络的训练过程。下面用两层神经网络为例来叙述 BP 网络的训练步骤。

步骤 1：用小的随机数对每一层的权值 w 和偏差 b 初始化，以保证网络不被大的加权输入饱和，同时还要进行以下参数的设定或初始化：

- 设定期望误差最小值：err_goal
- 设定最大循环次数：$\max_epoch$
- 设置修正权值的学习速率：一般选取 $lr=0.01\sim0.7$
- 从 1 开始的循环训练：$for_epoch=1:\max_epoch$

步骤 2：计算网络各层输出矢量 $A1$ 和 $A2$，以及网络误差 E：

$$A1 = \tan sig(w1 * p, b1);$$
$$A2 = purelin(w2 * A1, b2);$$
$$E = T - A;$$

步骤 3：计算各层反向传播的误差变化 $D2$ 和 $D1$，并计算各层权值的修正值及新的权值：

$$D2 = deltalin(A2, E);$$
$$D1 = deltalan(A1, D2, w2);$$
$$[dw1, db1] = learnbp(p, D1, lr);$$
$$[dw2, db2] = learnbp(A1, D2, lr);$$
$$w1 = w1 + dw1; b1 = b1 + db1;$$
$$w2 = w2 + dw2; b2 = b2 + db2;$$

步骤 4：再次计算权值修正后的误差平方和：

$$SSE = sumsqr(T - purelin(w2 * \tan sig(w1 * p, b1), b2));$$

步骤 5：检查 SSE 是否小于 err_goal，若是，则训练结束；否则继续。

以上就是 BP 网络利用 MATLAB 神经网络工具箱训练的过程。以上所有的学习规则与训练的全过程，还可以用函数 *trainbp* 来代替。它的使用同样需要定义有关参数：显示间隔次数、最大循环次数、目标误差和学习速率，在调用 *trainbp* 函数后，返回训练后的权值、循环训练的总数和最终误差。

3.2.5　BP 算法的改进

在实际应用中，原始的 BP 算法很难胜任，因此出现了很多的改进算法。BP 算法的改进主要有两种途径，一种是采用启发式学习方法，另一种则是采用更有效的优化算法。

在神经网络工具箱中，函数 *trainbpx*( ) 采用动量法和学习速率自适应调整两种策略，从而提高了学习速度并增加了算法的可靠性，动量法降低了网络对于误差曲面局部细节的敏感性，有效地抑制了网络陷于局部极小；自适应调整学习速率有利于缩短学习时间。

3.3　线性神经网络

线性神经网络是最简单的一种神经元网络，它可以由一个或多个线性神经元构成。20 世纪 50 年代末由 Widrow 和 Hoff 提出的自适应线性元件（Adaptive Linear Element，Adaline）是线性神经网络最早的典型代表。线性神经网络与感知器神经网络的不同之处在于其每个神经元的传递函数为线性函数，因此线性神经网络的输出可以取任意值，而感知器神经网络的输出只能是 1 或 0。线性神经网络可以采用 Widrow-Hoff 学习规则，也称为 LMS（Least Mean Squre）算法来调整网络的权值和阈值。线性神经网络的学习算法比感知器网络的学习算法的收敛速度和精度都有较大的提高。

线性神经网络主要用于函数逼近、信号处理滤波、预测、模式识别和控制等方面。

3.3.1 线性神经元模型

图 3-9 给出一个线性神经元模型，其传递函数为线性传递函数 purelin，如图 3-10 所示。

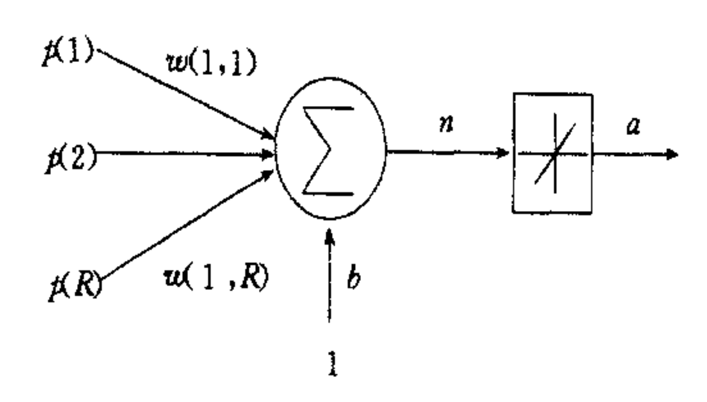


图 3-9 线性神经元模型

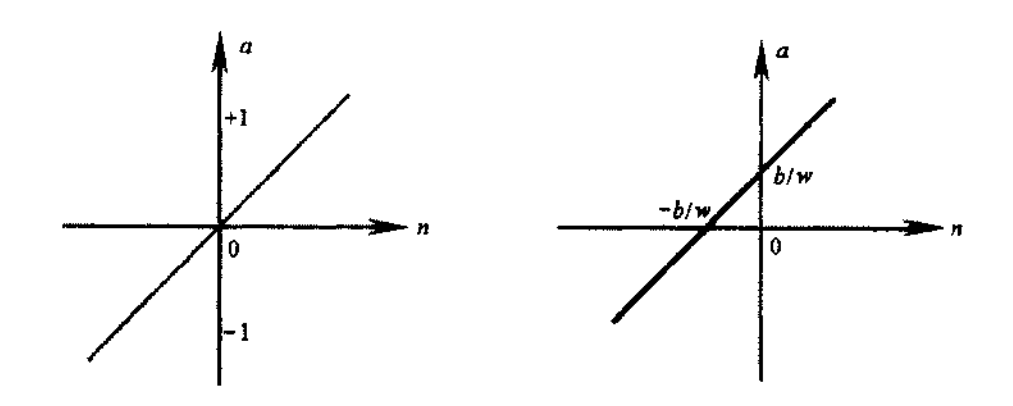


图 3-10 线性传递函数 purelin

由于线性神经网络中神经元的传递函数为线性函数，其输入输出之间是简单的比例关系。因此，对于单个线性神经元，可通过下式计算出：

$$a = purelin(w \times p + b)$$

在 MATLAB 6.5 中可用：

$$net = newlind(P,T)$$

$$a = sim(net, P)$$

生成和计算线性神经元的输出。

3.3.2　线性神经网络的模型

图 3-11 给出的是具有 R 个输入的单层（有 S 个神经元）线性神经元网络的两种形式，其权值为矩阵 $\boldsymbol{w}$，阈值为向量 $\boldsymbol{b}$，这种网络也称为 Madaline 网络。

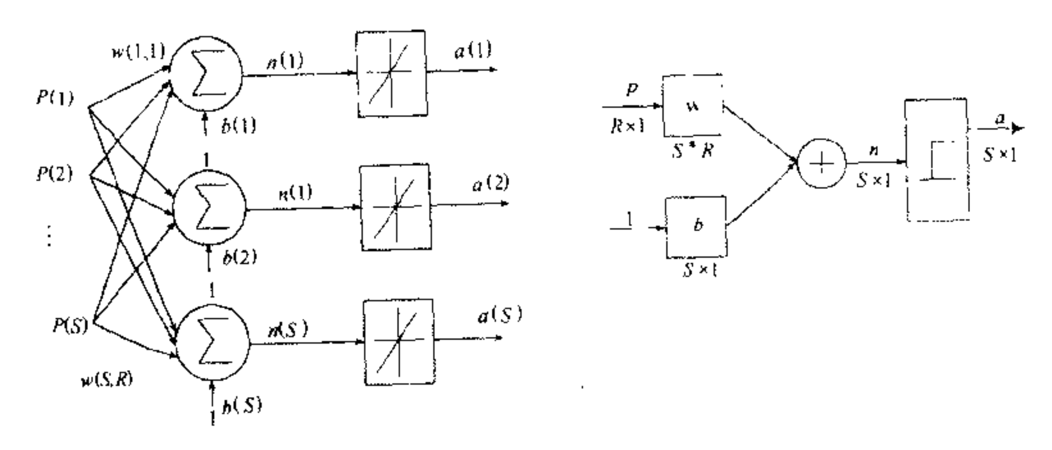


图 3-11　线性神经元网络

Widrow-Hoff 学习规则只能训练单层的线性神经元网络，但这一点并不影响单层线性神经网络的应用。因为对每一个多层线性神经网络而言，都可以设计出一个性能相当的单层线性神经网络。

3.3.3　线性网络的初始化

与感知器网络相同，线性网络的初始化也采用 MATLAB 6.5 神经网络工具箱中的 init( ) 函数。其用法为：

$$net = init(net);$$

这条语句可以将 net 网络权值和阈值设置为 0。MATLAB 神经网络工具箱中的函数 newlin 及 newlind 均可产生一个单层线性网络，在这两个函数中使用了 init( )，因此得到的线性网络已经是初始化的。

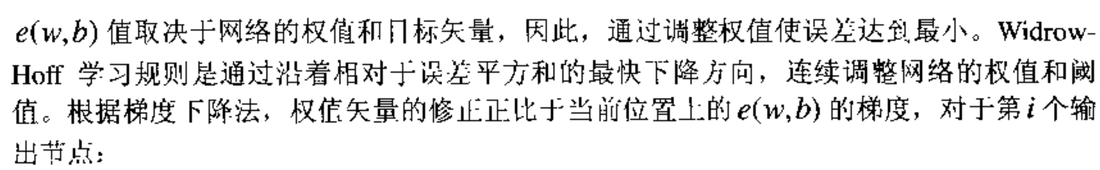
3.3.4　线性网络的学习规则

线性网络采用 Widrow-Hoff 学习规则，利用 learnwh()数来修正网络的权值和阈值。使用 Widrow-Hoff 学习规则可以用来训练网络某一层的权值和阈值，使其线性逼近一个函数式。

首先定义一个线性网络的误差函数：

$$e(w,b) = \frac{1}{2}[t-a]^2 = \frac{1}{2}[t-wp]^2$$

由上式可见，线性网络具有抛物面型的误差曲面，因此只能有一个误差最小值。由于



$e(w,b)$ 值取决于网络的权值和目标矢量，因此，通过调整权值使误差达到最小。Widrow-Hoff 学习规则是通过沿着相对于误差平方和的最快下降方向，连续调整网络的权值和阈值。根据梯度下降法，权值矢量的修正正比于当前位置上的 $e(w,b)$ 的梯度，对于第 i 个输出节点：

$$\Delta w(i,j) = -\eta \frac{\partial e}{\partial w(i,j)} = \eta[t(i)-a(i)]p(j)$$

或表示为：

$$\Delta w(i,j) = \eta\delta(i)p(j)$$
$$\Delta b(i) = \eta\delta(i)$$

其中 $\delta(i) = t(i) - a(i)$。

以上两组表达式即为 Widrow-Hoff 学习规则，又称为最小均方误差算法（LMS）。Widrow-Hoff 学习规则的权值变化量正比于网络的输出误差及网络的输入矢量。该算法无需求导数，因此比较简单，又具有收敛速度快和精度高的优点。上述式中 η 是学习速率，但学习速率较大时，学习过程加速，网络收敛较快；但是 η 太大时，学习过程变得不稳定，且误差会加大。因此学习速率的取值很关键。神经网络工具函数 maxlinlr 用于求合适的学习速率 lr（即 η）。网络没有阈值时，用 $lr = \max linr(P)$ 求学习速率；网络有阈值时，用 $lr = \max linr(P,'bias')$ 求学习速率。

采用 Widrow-Hoff 学习规则训练的线性网络，该网络能够收敛的必要条件式被训练的输入矢量必须是线性独立的，且应适当地选择学习速率。

3.3.5 线性网络的训练

线性网络的训练过程分如下 3 步：

（1）根据给定的输入矢量计算网络的输出矢量 $a = w \times p + b$，以及与期望输出之间的误差 $e = t - a$；

（2）将网络输出误差的平方和与期望误差相比较，如果其值小于期望误差，或训练已达到事先设定的最大训练次数，则终止训练；否则，继续训练；

（3）采用 Widrow-Hoff 学习规则计算新的权值和阈值，并返回到第（1）步。

如果网络训练成功，那么当一个不在训练中的输入矢量输入到网络中去时，网络产生一个与此相应的输出矢量。这种特性被称为泛化功能，这在函数逼近以及输入矢量分类的应用中是很有用的。

如果经过训练的网络不能达到期望目标，可以有两种选择：或者检查一下所要解决的问题是否适用于线性网络，或对网络进行进一步的训练。

在 MATLAB 6.5 的神经网络工具箱中，线性网络的训练函数为 adapt( ),adaptwb( )和 train( ),trainwb( )。

3.4 径向基函数网络

众所周知，BP 网络用于函数逼近时，权值的调节采用的是负梯度下降法，这种调节

权值的方法有它的局限性，即存在着收敛速度慢和局部极小等缺点。而径向基函数网络（Radial Basis Function，RBF）无论在逼近能力、分类能力和学习速度等方面均优于 BP 网络。

3.4.1　径向基函数网络模型

RBF 网络由三层组成，其结构如图 3-12 所示。输入层节点只是传递输入信号到隐层，隐层节点（也称 RBF 节点）由像高斯核函数那样的辐射状作用函数构成，而输出层节点通常是简单的线性函数。

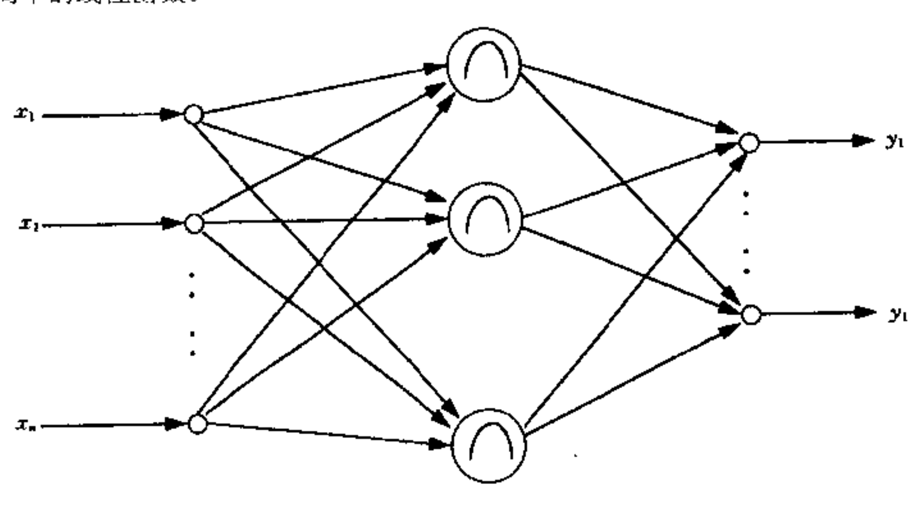


图 3-12　RBF 网络

隐层节点中的作用函数（核函数）对输入信号将在局部产生响应，也就是说，当输入信号靠近核函数的中央范围时，隐层节点将产生较大的输出。由此可看出这种网络具有局部逼近能力，所以径向基函数网络也称为局部感知场网络。

3.4.2　基函数的形式

作为基函数的形式，有下列几种：

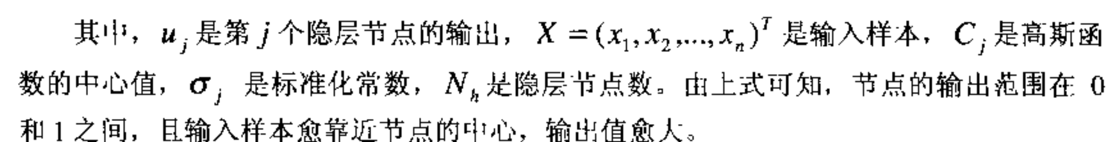
$$f(x)=\exp^{-(x/\delta)^2}$$

$$f(x)=\frac{1}{(\delta^2+x^2)^a}\quad ,a>0$$

$$f(x)=(\delta^2+x^2)^{\beta}\quad ,a<\beta<1$$

上面这些函数都是径向对称的，虽然有各种各样的核函数，但最常用的是高斯核函数（Gaussian kernel function），如下式所示：

$$u_j=\exp\left[-\frac{(X-C_j)^T(X-C_j)}{2\sigma_j^2}\right]\qquad j=1,2,\ldots,N_h$$



其中，u_j 是第 j 个隐层节点的输出，$X=(x_1,x_2,...,x_n)^T$ 是输入样本，C_j 是高斯函数的中心值，σ_j 是标准化常数，N_h 是隐层节点数。由上式可知，节点的输出范围在 0 和 1 之间，且输入样本愈靠近节点的中心，输出值愈大。

采用高斯基函数，具备如下优点：

- 表示形式简单，即使对于多变量输入也不增加太多的复杂性。
- 径向对称。
- 光滑性好，任意阶导数存在。
- 由于该基函数表示简单且解析性好，因而便于进行理论分析。

考虑到提高网络精度和减少隐层节点数，也可以将网络基函数改成多变量正态密度函数：

$$R_i(x)=\exp(-\frac{1}{2}(x-c_i)K(x-c_i)^T$$

其中 $K=E[(x-c_i)^T(x-c_i)^{-1}$ 是输入协方差阵的逆。注意这时上式已不再是径向对称。

3.4.3 RBF 学习过程

RBF 网络的输出为隐层节点输出的线性组合，即：

$$y_i=\sum_{j=1}^{N_h} w_{ij}u_j-\theta=W_i^T U \qquad i=1,2,...,m$$

其中 $W_i=(w_{i1},w_{i2},...w_{iN_h},-\theta)^T$，$U=(u_1,u_2,...u_{N_h},1)^T$。

RBF 网络的学习过程分为两个阶段。第一阶段，根据所有的输入样本决定隐层各节点的高斯核函数的中心值 C_j 和标准化常数 σ_j。第二阶段，在决定好隐层的参数后，根据样本，利用最小二乘原则，求出输出层的权值 W_i。有时在完成第二阶段的学习后，再根据样本信号，同时校正隐层和输出层的参数，以进一步提高网络的精度。

从理论上而言，RBF 网络和 BP 网络一样可近似任何的连续非线性函数。两者的主要差别在于各使用不同的作用函数，BP 网络中的隐层节点使用的是 Sigmoid 函数，其函数值在输入空间中无限大的范围内为非零值，而 RBF 网络中的作用函数则是局部的。

3.5 GMDH 网络

GMDH（The Group Method of Data Handling）网络是前向网络的一种，称为多项式网络。它成功地应用于非线性系统的建模和控制中，如超音速飞机的控制系统，电力系统的负荷预测等。

图 3-13 所示的是一种典型的 GMDH 网络，它由 4 个输入和单输出构成。输入层节点只是传递输入信号到中间隐层的节点，每一隐层节点和输出节点正好有两个输入，因此单

输出节点的前一层肯定只有两个隐层节点。

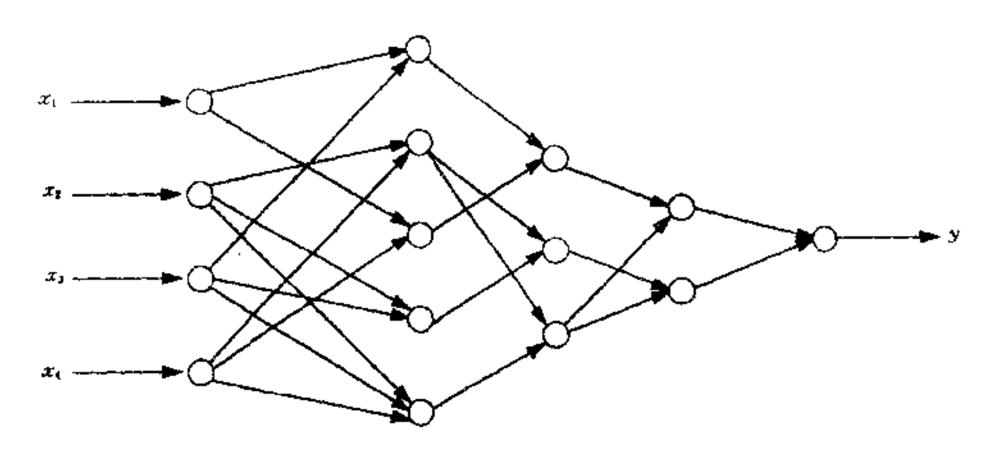


图 3-13 GMDH 典型网络结构

除输入层外，每个处理单元具有图 3-14 所示的形式，其输入输出关系如下：

$$z_{k,l} = a_{k,l}(z_{k-1,i})^2 + b_{k,l}z_{k-1,i}z_{k-1,j} + c_{k,l}(z_{k-1,j})^2 + d_{k,l}z_{k-1,i} + e_{k,l}z_{k-1,j} + f_{k,l}$$

其中 $z_{k,l}$ 表示第 k 层的第 l 个处理单元，且 $z_{0,l} = x_l$。由上式可见，GMDH 网络中的处理单元的输出是两个输入量的 2 次多项式，因此网络的每一层将使得多项式的次数增大 2 阶，其结果是网络的输出可以表示成输入的高阶（$2k$ 阶）多项式，其中 k 是网络的层数（不含输入层）。

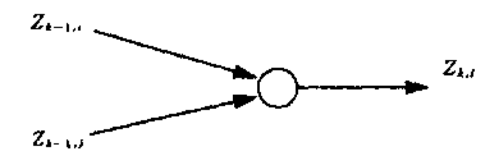


图 3-14 GMDH 网络的处理单元

下面将如何决定 GMDH 网络的每一隐层及处理单元的方法简述如下：

- 首先在输入层厚加上一层中间层，决定其处理单元的数目和每一处理单元的多项式表达式。
- 同样在中间层后，再加上一层中间层，做相同的处理，一直到输出层。每一层的处理内容和方法都相同。

第 4 章　前向型神经网络设计分析

前向型神经网络在网络结构上采用的是其信息只能从输入层单元传递到它后面一层的单元的方式。其结构在前面已经进行了介绍。这种结构的神经网络通常比较适合于预测、模式识别，以及非线性函数逼近。前面介绍了前向型神经网络的理论并列举了一些实例，本章将介绍前向型神经网络的设计方法并。

本章主要内容包括：

- 感知器神经网络设计
- 线性网络设计
- 自适应预测
- 线性系统辨识
- 自适应系统辨识
- 函数逼近
- 胆固醇含量估计
- 特征识别
- 径向基函数网络设计

4.1 引　　言

前向型神经网络在网络结构上采用的是其信息只能从输入层单元传递到它后面一层的单元的方式。其结构在前面已经进行了介绍。这种结构的神经网络通常比较适合于预测、模式识别，以及非线性函数逼近。

前向型神经网络包括感知器神经网络、线性神经网络、BP（Backpropagation）网络和径向基函数网络。

感知器神经网络通常用于简单的模式分类中。对于线性可分的几类模式，感知器神经网络可以成功地对它们进行分类。但是，对于线性不可分的情况，感知器神经网络不能有效地分类。例如，对于“异或”问题，感知器神经网络就不能解决。本章的第 2 节将介绍运用感知器神经网络进行模式分类设计的实例。

线性神经网络与感知器神经网络的不同之处在于其每个神经元的传递函数为线性函数，因此线性神经网络的输出可以取任意值，而感知器神经网络的输出只能是 0 或者 1。线性神经网络主要用于函数逼近、信号预测、系统辨识、控制以及模式识别等领域。本章的第 3 节将介绍一个利用线性网络进行信号预测的实例；第 4 节将介绍一个利用线性网络进行自适应信号预测的实例；第 5 节将介绍一个利用线性网络进行线性系统辨识的实例；第 6 节将介绍一个利用线性神经网络进行自适应系统辨识的实例。

在实际应用中，我们用得最广泛的是反向传播网络（BP 网络）。一个经过训练的 BP

网络能够根据输入给出合适的结果，哪怕这个输入并没有被训练过。这个特性使得 BP 网络很适合采用输入/目标对来进行训练，而且并不需要把所有可能的输入/目标对都拿来训练。本章的第 7 节将介绍一个利用 BP 网络进行非线性网络逼近的实例；第 8 节将介绍一个在医疗中使用 BP 网络估计胆固醇含量的实例；第 9 节将介绍一个利用 BP 网络进行特征分类，从而对字母进行识别的实例。

还有一种很重要的前向型神经网络，即径向基函数神经网络。这种网络是一种局部逼近网络，它在很多方面都优于 BP 网络。如它有很强的分类能力和逼近能力，在学习速度方面也比 BP 网络要快。在本章的第 10 节将介绍一个利用径向基函数神经网络进行函数逼近的实例。

4.2 感知器神经网络设计

使用感知器神经网络可以进行简单的分类器设计。当它用于两类模式时，相当于在高维样本空间中，用一个超平面将两类样本分开。Rosenblatt 和其他一些人曾经给出过严格的数学证明，对于线性可分的样本，算法是收敛的，即能够找到合适的权值。而对于线性不可分的样本，判定界面会产生振荡，以至于权值不收敛。

下面我们用函数设计一个非常简单的 2 输入感知器神经网络，利用它来实现分类器的功能，并对它进行一些讨论。

图 4-1 给出了感知器神经网络的结构图。

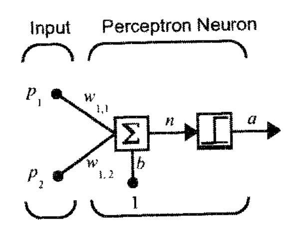


图 4-1 感知器神经网络结构

4.2.1 问题描述

首先给出一些已知的样本点，假设它们的坐标值和类别都已确定。我们设计一个感知器神经网络，并利用这些已知的样本点来训练网络，使得对于以后任意给定一些点，感知器神经网络都能尽可能正确地将它们分类。



已知给出的样本点及其类别，在坐标图中把它们表示出来，不同的类别使用了不同的符号，如图 4-2 所示。

```
P=[-0.5 -0.5   0.3 -0.1 0.2   0.0 0.6   0.8;
-0.5   0.5 -0.5   1.0 0.5 -0.9 0.8 -0.6];
T=[1 1 0 1 1 0 1 0 ];
plotpv(P,T);
```

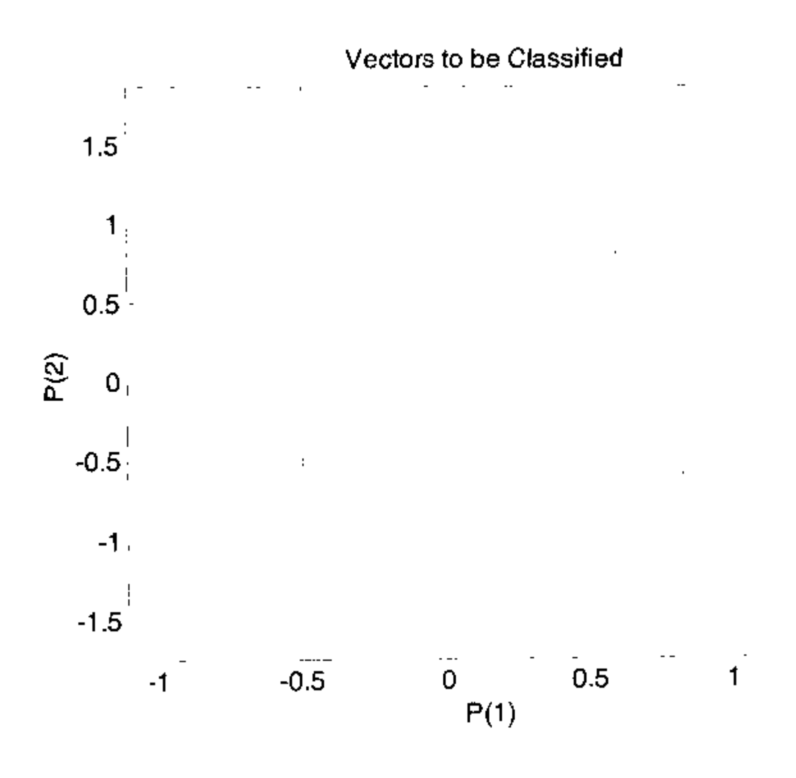


图 4-2　样本点的分布及其相应的类别

4.2.2　网络初始化

使用 newp 函数建立一个感知器神经网络。该函数执行后返回一个新的感知器神经网络，传递函数在默认情况下为 hardlim 函数，学习函数在默认情况下为 learnp 函数。我们的样本点取值在－1～1 之间，而网络只有单个神经元，所以有：

```
net=newp([-1 1;-1 1],1);
```

使用 plotpc 函数可以在已绘制的图上加上感知器分类线。我们让它返回得到的分类线的句柄，以便在下一次再绘制分类线时能够将旧的删除。

```
handle=plotpc(net.iw{1},net.b{1});
```

4.2.3　网络训练

神经网络建立好以后，还必须经过训练才能够实际应用，通过训练，以决定网络的权

值和阈值。对于感知器网络来说，其训练过程为：

对于给定的输入向量，计算网络的实际输出，并与相应的目标向量进行比较，得到误差 e，然后根据相应的学习规则调整权值和阈值。重新计算网络在新的权值和阈值作用下的输出，重复上述的权值和阈值的调整过程，直到网络的输出与期望的目标向量相等或者训练次数达到预定的最大次数时才停止训练。之所以要设定最大训练次数，是因为对于有些问题，使用感知器神经网络时是不能解决的，这正是感知器神经网络的缺陷。

在这里，使用 adapt 函数来训练网络，调整网络的权值和阈值。

```
E=1;
while (sse(E))
[net,Y,E]=adapt(net,P,T);
handle=plotpc(net.iw{1},net.b{1},handle);
end;
```

代码中使用了 sse 函数得到网络的平方和误差，以此作为一项重要的性能指标参数。图 4-3 显示了网络在进行训练以后得到的分类线。从图 4-3 中可见分类线能够将两类输入向量进行正确的分类。

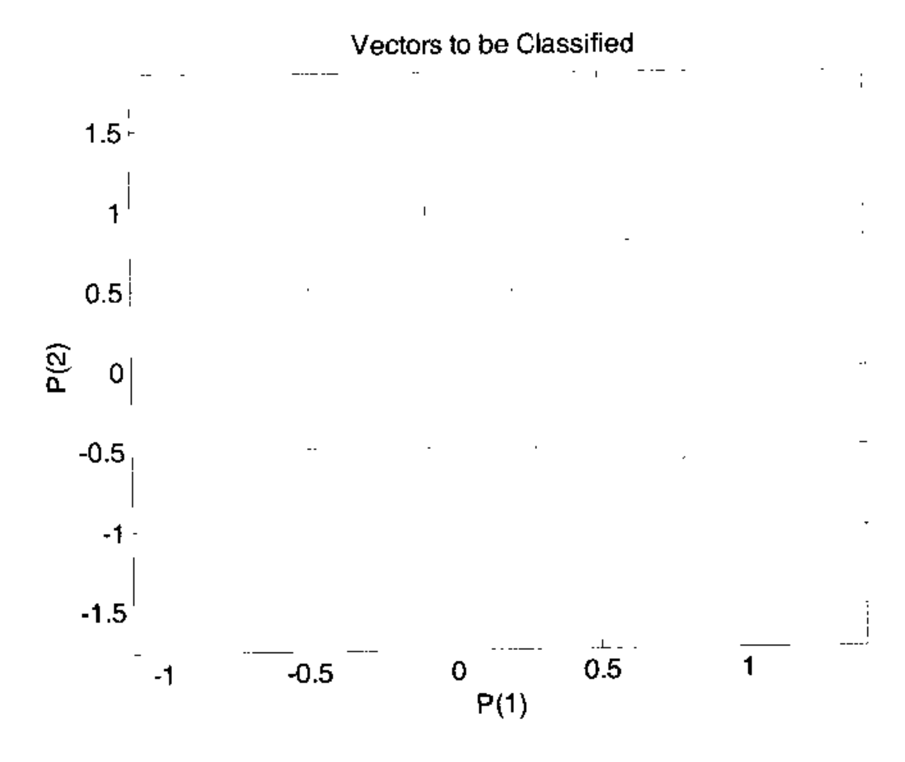


图 4-3　网络设计结果——分类线

4.2.4　神经网络性能测试

上面已经完成了网络的训练。一旦训练完成，神经网络的权值和阈值就已经确定了，

于是就可以使用它来解决实际的问题。接下来我们选择一些点，输入到网络中，利用其得到的结果测试其性能。

选择了10个点，其坐标值见如下代码：

```
testpoints=[-0.5  0.3 -0.9  0.4 -0.1  0.2 -0.6  0.8  0.1 -0.4;
-0.3 -0.8 -0.  .7  0.4 -0.6  0.1 -0.5 -0.5  0.3];
a=sim(net,testpoints);
figure;
plotpv(testpoints,a);
plotpc(net.iw{1},net.b{1},handle);
```

得到了网络对这些测试点的输出值，并且相应的在图中做出了标记（两类点分别用不同的记号表示）。分类结果如图 4-4 所示。从测试的结果来看，网络能够将它们正确地分类。这也说明了设计的网络是正确的。

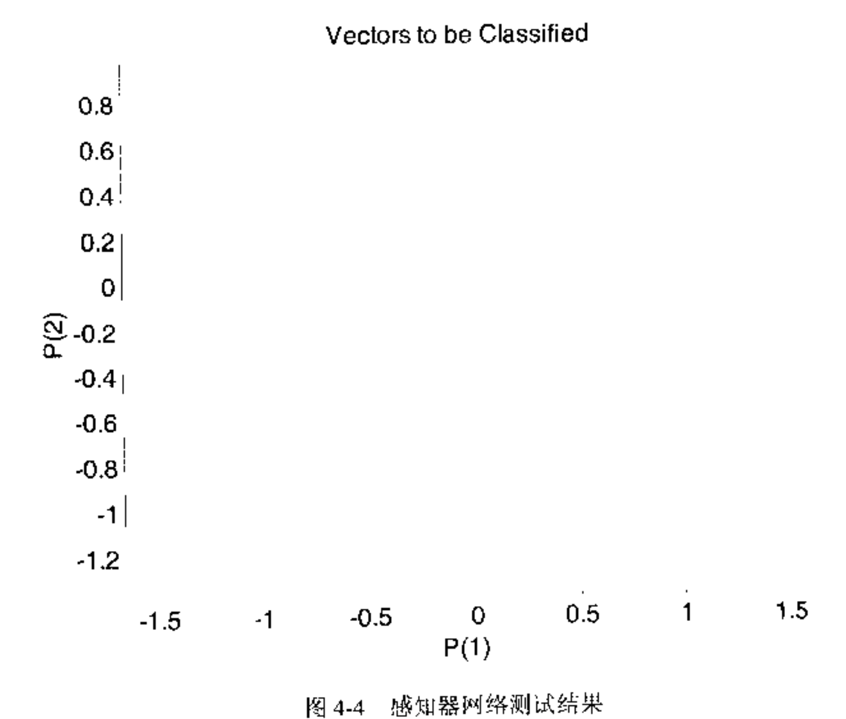


图 4-4 感知器网络测试结果

例程 4-1 是上述设计的 MATLAB 程序代码。

例程 4-1

```
%输入样本点及其相应的类别
P = [-0.5  -0.5   0.3  -0.1  0.2   0.0   0.6  0.8;
-0.5   0.5  -0.5   1.0  0.5  -0.9  0.8  -0.6];
T = [1 1 0 1 1 0 1 0 ];
%在坐标图上绘出样本点
plotpv(P,T);
%建立一个感知器网络
net=newp([-1 1;-1 1],1);
```

```
handle=plotpc(net.iw{1},net.b{1});
%利用样本点训练网络并绘出得到的分类线
E=1;
while (sse(E)),
[net,Y,E]=adapt(net,P,T);
handle=plotpc(net.iw{1},net.b{1},handle);
end;
%选择 10 个点来测试网络
testpoints=[-0.5   0.3 -0.9   0.4 -0.1   0.2 -0.6   0.8   0.1 -0.4;
-0.3 -0.8 -0.4 -0.7   0.4 -0.6   0.1 -0.5 -0.5   0.3];
a=sim(net,testpoints);
%在坐标图上绘出网络的分类结果及分类线
figure;
plotpv(testpoints,a);
plotpc(net.iw{1},net.b{1},handle);
```

4.2.5 结论及讨论

1．结论

我们使用感知器神经网络设计了一个简单的分类器，并使用已知的样本点对其进行了训练。该分类器能够实现两类模式的分类。在网络训练好以后，选择了一些点，把它们输入了这个网络，网络给出了正确的分类结果。

但是，这个感知器神经网络的设计过程是比较脆弱的。如果给出的问题是线性不可分的，或者是样本中存在奇异点，网络的设计都不会如此顺利。

实际上，感知器神经网络在结构和学习规则上都有很大的局限性，这些局限性使得其应用被局限在一定的范围内。感知器神经网络的局限性包括：

- 函数是阈值函数，所以感知器神经网络的输出只能取 0 或 1，因此，感知器只能用于简单的分类问题。
- 当感知器神经网络的输入样本中存在奇异样本点，即该样本点向量同其他所有样本点向量相比特别大或者特别小时，网络的训练将要花费很长的时间。
- 感知器神经网络只能对线性可分的向量集合进行分类。前面已经提到过，Rosenblatt 和其他一些人曾经给出过严格的数学证明，对于线性可分的样本，感知器在有限的时间内总能达到目标向量。而对于线性不可分的样本，判定界面会产生振荡，以至于权值不收敛。而且，如何确定输入向量是否线性可分，是很困难的，至今也没有非常有效的办法，尤其当输入向量增多时，更难以确定。一般只有设定一定的循环次数，对网络进行训练而判定其是否线性可分。

2．讨论

下面针对刚才提到的感知器神经网络局限性的第二条进行讨论，即当输入样本中存在奇异样本点的情况。

模型与前面设计的类似，只是输入样本点有了变化，在原来的基础上加入了一个奇异

点（60，20），与其他样本值相比，其坐标值非常大。

```
P=[-0.5 -0.5   0.3 -0.1 0.2   0.0 0.6   0.8 60;
-0.5   0.5 -0.5   1.0 0.5 -0.9 0.8 -0.6 20];
T=[1 1 0 1 1 0 1 0 1];
```

得到输入向量的分布情况，与前面一样，这些样本点都被加上了相应的标记，以区分类别。如图 4-5 所示，显然，有一个点明显地与其他点之间有很大的距离。

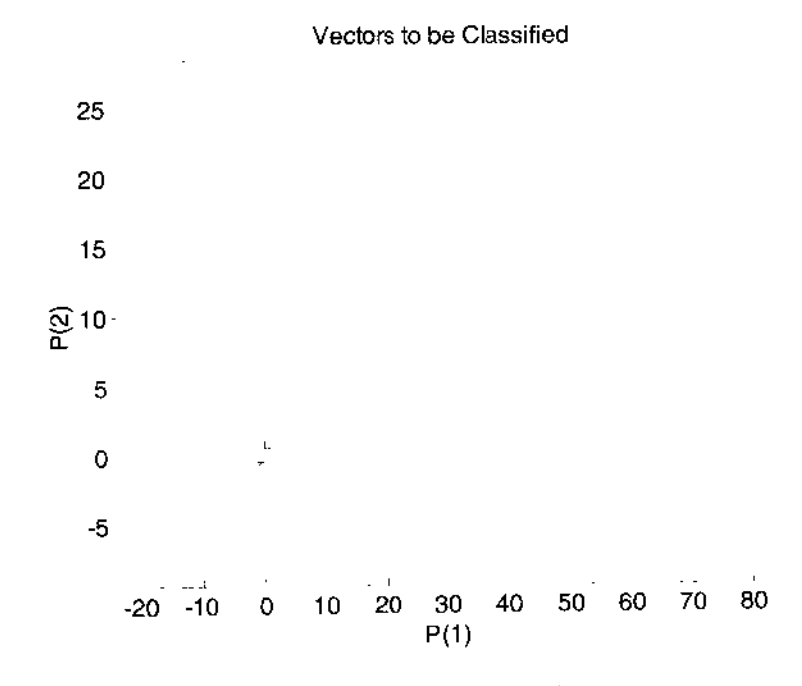


图 4-5　含有奇异点的输入样本图

用这些样本点对前面设计的感知其神经网络进行训练。由于输入点的坐标分布发生了变化，故建立网络的代码相应改变为：

```
net=newp([-1 60;1 20],1);
```

与前面的方法一样，对建立的网络进行训练，以得到合适的权值和阈值。这部分代码没有什么变化。训练完成以后，在图上绘出了得到的分类线。

由于奇异样本点的存在，可以发现，感知器神经网络要花费比前面多得多的时间来完成训练过程。

图 4-6（a）中显示了训练后得到的分类线。由于奇异样本点的存在，使得原始样本点在图中聚集在一个很小的范围内，从这个图上很难看出分类线对它们的分类能力。我们将它们所在的局部进行放大，在图 4-6（b）中重新显示了这个分类线。下面的代码完成了局部放大显示的功能。

```
figure;
plotpv(P,T);
plotpc(net.iw{1},net.b{1});
axis([-2 2 -2 2]);
```

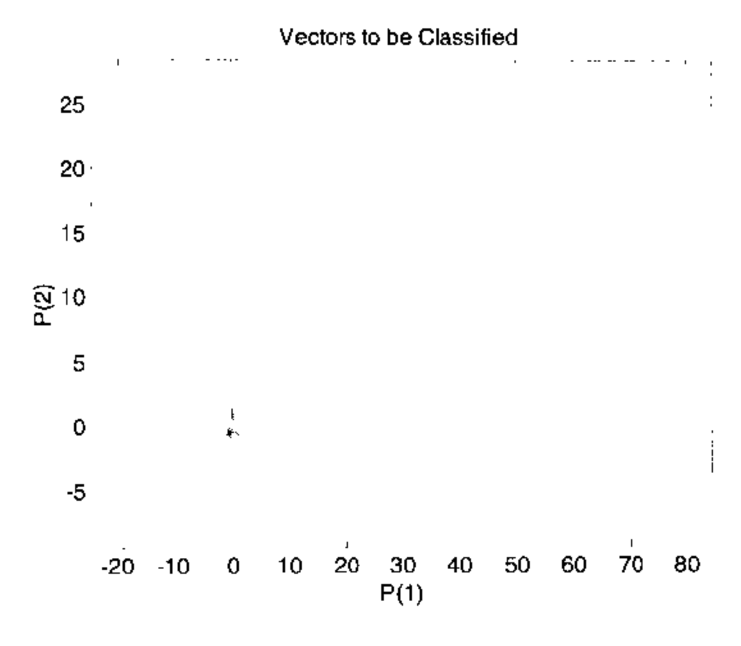


图 4-6（a） 训练得到的分类线

图 4-6（b）即为将局部放大后得到的分类情况。从图中的分类线图可以看出，在两类样本点中均有一个点与分类线十分接近，但还是得到了正确的结果。从这一点来看，也可以知道，样本中含有奇异点对整个网络的训练影响是很大的。

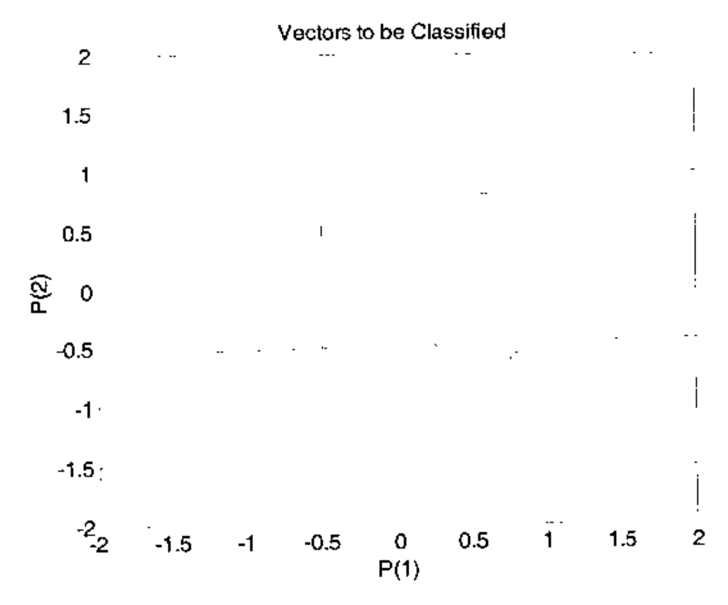


图 4-6（b） 训练得到的分类线（局部放大后的效果）

设计好网络后，要测试其性能。与前面一样，选择一些测试点，使用设计好的网络对它们进行分类，在图中相应的对分类结果做出标记，并将分类线也绘在图上，观察标记情况与实际的分布是否吻合，以此来检验网络的正确性。这里我们选择与前面一样的点进行

测试，代码也没有什么变化，如图 4-7 所示。

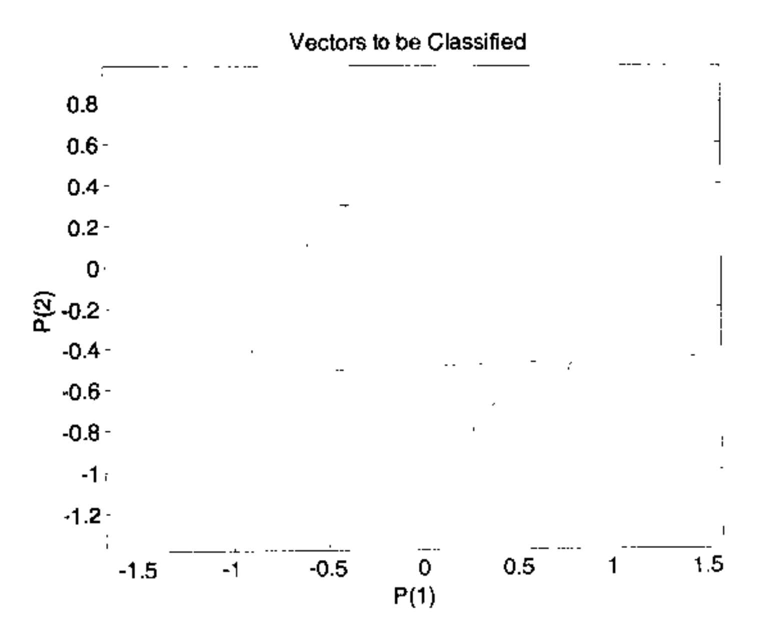


图 4-7　测试结果

图 4-7 中显示了测试点及分类线。可以看出，网络对它们进行了正确的分类。但是，需要注意的是，有几个点非常靠近分类线，这使得在实际中这个网络的容错能力不强，很有可能出现错误。例程 4-2 是相应的 MATLAB 程序代码。

例程 4-2

```
%输入样本点及其相应的类别,其中有一个奇异点
P=[-0.5  -0.5   0.3  -0.1  0.2   0.0   0.6  0.8  60;
-0.5   0.5  -0.5   1.0  0.5  -0.9  0.8  -0.6 20];
T=[1 1 0 1 1 0 1 0 1];
%在坐标图上绘出样本点
plotpv(P,T);
%建立一个感知器网络
figure;
plotpv(P,T);
net=newp([-1 60;1 20],1);
handle=plotpc(net.iw{1},net.b{1});
%利用样本点训练网络并绘出得到的分类线
E=1;
while (sse(E)),
[net,Y,E]=adapt(net,P,T);
handle=plotpc(net.iw{1},net.b{1},handle);
end;
%局部放大分类线图
figure;
```

```
plotpv(P,T);
plotpc(net.iw{1},net.b{1});
axis([-2 2 -2 2]);
%选择 10 个点来测试网络
testpoints=[-0.5  0.3 -0.9  0.4 -0.1  0.2 -0.6  0.8  0.1 -0.4;
-0.3 -0.8 -0.4 -0.7  0.4 -0.6  0.1 -0.5 -0.5  0.3];
a=sim(net,testpoints);
%在坐标图上绘出网络的分类结果及分类线
figure;
plotpv(testpoints,a);
plotpc(net.iw{1},net.b{1});
```

对于线性不可分的情况，在训练时是得不到正确的分类线的。读者可以自己试着用前面的方法进行讨论，限于篇幅，本处就不再赘述了。

4.3 利用线性网络进行信号预测

线性神经网络在函数逼近、信号处理滤波、预测、模式识别和控制等方面都有广泛的应用，本节我们将详细介绍它的设计方法。

4.3.1 问题描述

首先定义一个信号 T，它是一个采样速率为每秒 40 次，持续 5 秒钟的正弦信号。

```
Time=0:0.025:5;
T=sin(Time*4*pi);
Q=length(T);
```

在任意给定的时刻，给网络的输入为信号 T 最近的五个值，我们希望网络能根据这五个值输出下一个值。

可以用将信号 T 延迟 1～5 个时间步长的方法得到网络的输入 P。

```
P=zeros(5,Q);
P(1,2:Q)=T(1,1:(Q-1));
P(2,3:Q)=T(1,1:(Q-2));
P(3,4:Q)=T(1,1:(Q-3));
P(4,5:Q)=T(1,1:(Q-4));
P(5,6:Q)=T(1,1:(Q-5));
plot(Time,T);
title('信号 T');
xlabel('时间');
ylabel('目标信号');
```

绘出信号 T 的曲线如图 4-8 所示。

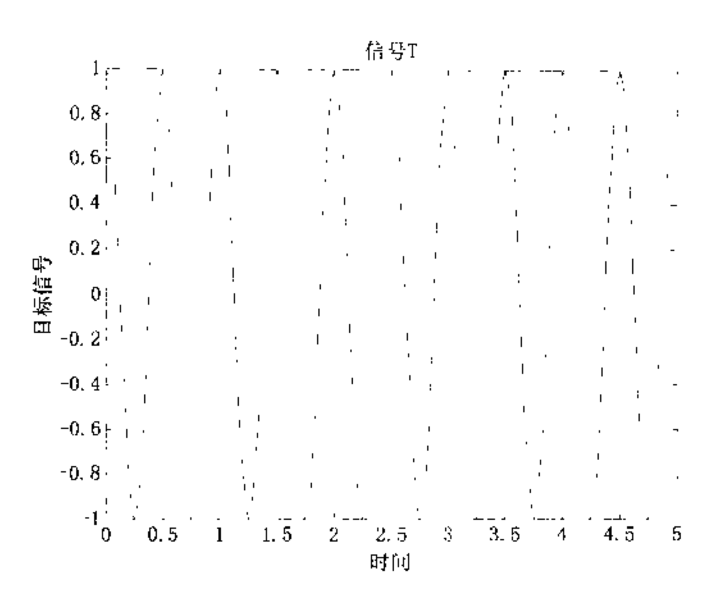


图 4-8 信号 T

4.3.2 网络设计

对于某一个固定信号来说，其过去和未来的值之间的关系将不会改变。因此，这个预测网络可以直接使用 newlind 函数来设计。本问题中有五个输入（即五个延迟信号的值）和一个输出（即预测的下一个时刻信号的值）。所以，所设计的网络必须含有一个具有五个输入的单神经元。网络的结构如图 4-9 所示。

在本问题中，利用 newlind 函数设计网络，可以得到网络的权重和阈值，并且使得误差平方和最小。

net=newlind(P,T);

网络设计好以后，下面就可以开始测试其性能了。

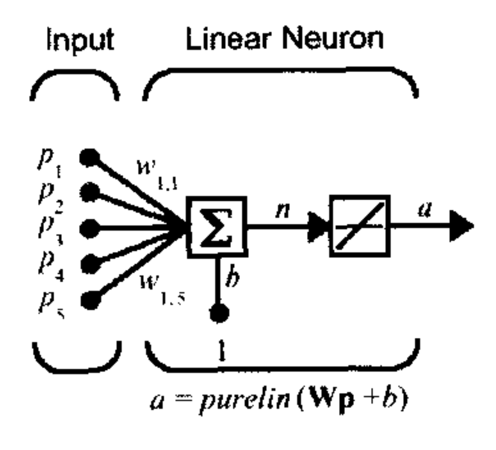


图 4-9 网络结构



4.3.3 测试网络

对于用上述方法设计的网络，测试其性能。方法是将前面产生的五个延迟信号 P 作为测试输入，利用仿真函数 sim 得到由网络运算的结果 a，如图 4-10 所示。

```
a=sim(net,P);
%绘出网络预测输出
figure;
plot(Time,a);
title('预测结果');
xlabel('时间');
ylabel('预测值');
```

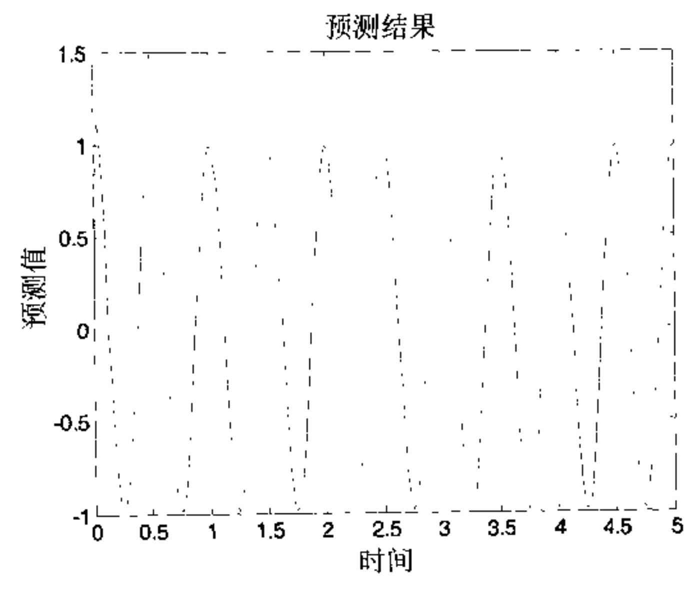


图 4-10　网络预测输出

比较图 4-8 的期望信号 T 曲线和图 4-10 得到的网络实际输出信号 a 曲线，可以发现它们十分接近。为了便于比较两者的差别，可以求出网络预测误差 e=T－a，并将其曲线绘出，如图 4-11 所示。

```
e=T-a;
figure;
plot(Time,e);
title('误差');
xlabel('时间');
ylabel('误差值');
```

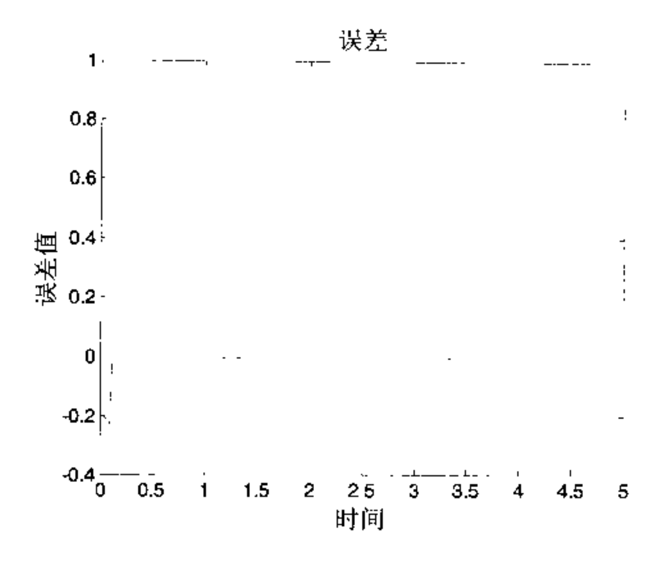


图 4-11　网络预测误差

观察误差曲线可以发现，在刚开始的一段时间里，网络误差较大，但经过很短的一段时间以后，误差逐渐减小，并趋于 0。这是由于在开始的几个时间步长里，网络输入信号不完整，而其需要的输入为五个信号，经过一段时间后，网络的输入变得完整了，因而其输出的误差也相应地减小。例程 4-3 是本问题的 MATLAB 程序代码。

例程 4-3

```
%生成一个信号，作为被预测信号
Time=0:0.025:5;
T=sin(Time*4*pi);
Q=length(T);
%由信号 T 生成输入信号 P
P = zeros(5,Q);
P(1,2:Q) = T(1,1:(Q-1));
P(2,3:Q) = T(1,1:(Q-2));
P(3,4:Q) = T(1,1:(Q-3));
P(4,5:Q) = T(1,1:(Q-4));
P(5,6:Q) = T(1,1:(Q-5));
%绘出信号 T 的曲线
figure;
plot(Time,T);
title('信号 T');
xlabel('时间');
ylabel('目标信号');
%设计网络
net=newlind(P,T);
%仿真网络
a=sim(net,P);
%绘出网络预测输出
```

```
figure;
plot(Time,a);
title('预测结果');
xlabel('时间');
ylabel('预测值');
%得到误差信号，并绘出其曲线
e=T-a;
figure;
plot(Time,e);
title('误差');
xlabel('时间');
ylabel('误差值');
```

4.3.4 结论

虽然使用线性网络不能够无误差地解决非线性问题，但是它能够在保证误差平方和最小的意义下逼近非线性问题的解。虽然这个解不是最优的，但在很多情况下，它能够用于非线性问题的解决，得到需要的结果。给定线性网络的信息越多，它就能越好地逼近非线性问题，得到的解的误差越小。

当然，如果要解决的问题非线性程度太高（太难用线性方法逼近），或者对结果的误差要求非常高（要求很小的误差），那么用这种线性网络的方法就比较困难了，这时使用BP 网络或者径向基函数网络的方法更适合一些。

4.4 自适应预测

在这个应用中，我们使用 adapt 函数来在线的训练一个线性网络，用它来预测一个时变信号序列。由于网络是在线的进行自适应训练的，因此它能够实时地跟踪变化的信号，即能够用于预测时变信号序列。

4.4.1 问题描述

需要预测的信号是一个持续时间为 6 秒的信号。在前 4 秒信号的采样频率为 20 赫兹，而从第 4 秒到第 6 秒的两秒里，信号的采样频率加倍，即为 40 赫兹。在 MATLAB 中可用如下代码得到这个信号 T：

```
Time1=0:0.05:4;
Time2=4.05:0.024:6;
Time=[Time1 Time2];
T=[cos(Time1*4*pi) cos(Time2*8*pi)];
T=con2seq(T);
Plot(Time,cat(2,T{:}));
```

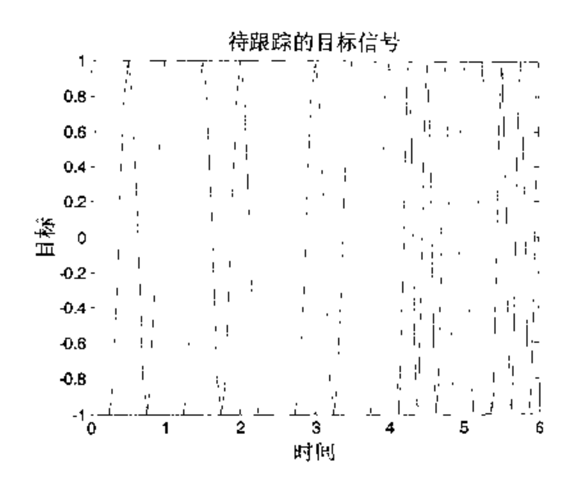


图 4-12　目标信号 T

信号 T 的分布如图 4-12 所示。

在本问题中，网络的输出是为了跟踪输入，所以网络的输入信号与目标信号相同。即有：

```
P=T;
```

4.4.2　网络初始化

在本问题中采用只有 1 个神经元的单输出线性网络，该网络有 5 个输入，分别为信号 T 的 5 个延迟信号。

使用 newlin 函数产生的网络结构如图 4-13 所示。设线性网络的学习速率为 0.1，于是可以用以下代码生成该网络。

```
lr=0.1;
delays = [1 2 3 4 5];
net = newlin(minmax(cat(2,P{:})),1,delays,lr);
```

其中 lr 为网络的学习速率，minmax(cat(2,P{:}))得到输入 P 的最大值、最小值矩阵，delays 为输入延迟向量。

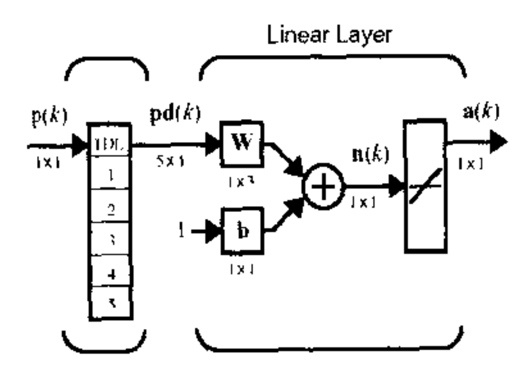


图 4-13　网络结构

4.4.3 网络训练

使用 adapt 函数对本网络进行自适应训练。在网络的输入为 P，输出目标为 T 的情况下，其代码如下。

```
[net,a,e]=adapt(net,P,T);
```

其中，a 为网络的输出，e 为网络的误差。

4.4.4 网络性能测试

在网络训练结束后，可以将其输出的预测信号与目标信号绘在同一幅图中进行比较。其结果如图 4-14 所示。

```
plot(Time,cat(2,a{:}),Time,cat(2,P{:}),'--');
xlabel('时间');
ylabel('目标、预测值');
title('目标信号及预测结果');
```

观察图 4-14 的曲线可知，在最开始，网络花了 1.5 秒（30 个采样点）的时间来跟踪目标信号，这段时间的误差较大。随后，网络的预测值能够能准确地跟踪目标信号。但是到了第 40 秒时，目标信号的频率发生了突变，这又使得预测值稍稍偏离了目标值。然而很快，由于网络的自适应学习功能，使得网络又对新的信号行为有了学习，输出信号又很快地跟踪到了目标信号。

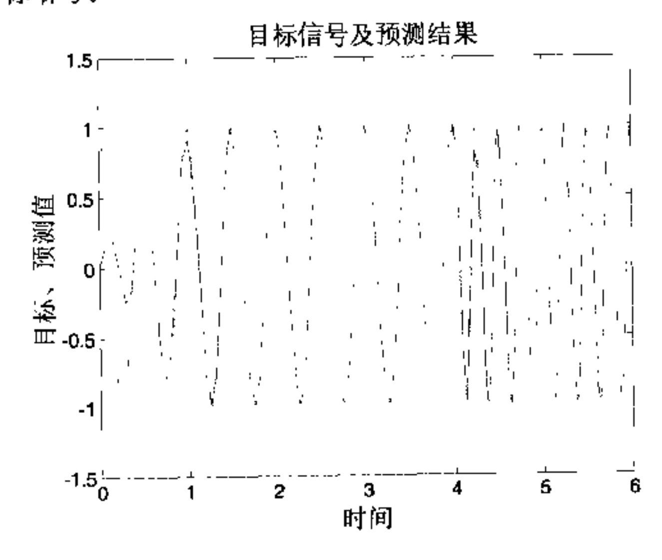


图 4-14 预测信号和目标信号的比较

为了更加直观地观察输出预测信号与目标信号的差别，可以绘出误差曲线，如图 4-15 所示。

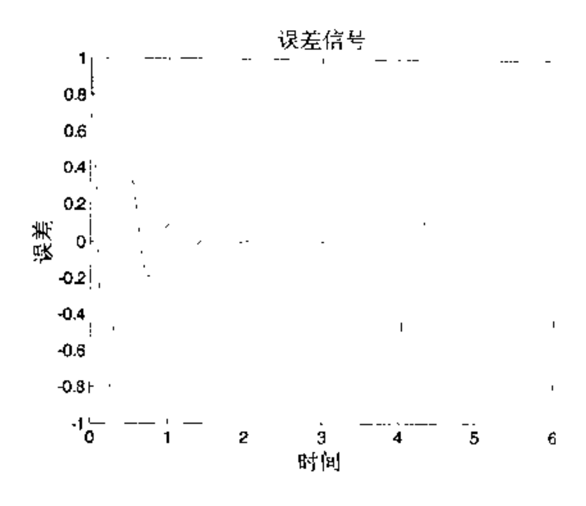


图 4-15　预测误差曲线

```
plot(Time,cat(2,e{:}));
xlabel('时间');
ylabel('误差');
title('误差信号');
```

例程 4-4 是本问题的 MATLAB 程序。

例程 4-4

```
%分别定义两段时间 Time1 和 Time2，对应信号的不同频率时段
Time1=0:0.05:4;
Time2=4.05:0.024:6;
Time=[Time1 Time2];
%得到待预测的目标信号
T=[cos(Time1*4*pi) cos(Time2*8*pi)];
T=con2seq(T);
%绘出目标信号的曲线，并指定给输入
figure;
plot(Time,cat(2,T{:}));
xlabel('时间');
ylabel('目标');
title('待跟踪的目标信号');
P=T;
%生成线性网络
lr=0.1;
delays = [1 2 3 4 5];
net = newlin(minmax(cat(2,P{:})),1,delays,lr);
%对网络进行自适应训练
[net,a,e]=adapt(net,P,T);
%绘出预测信号、目标信号及误差信号曲线
figure;
plot(Time,cat(2,a{:}),Time,cat(2,P{:}),'--');
```

```
xlabel('时间');
ylabel('目标、预测值');
title('目标信号及预测结果');
figure;
plot(Time,cat(2,e{:}));
xlabel('时间');
ylabel('误差');
title('误差信号');
```

4.4.5 结论

在本问题中设计的线性网络能够很快地适应目标信号的改变以对其进行跟踪。给人印象非常深刻的是网络的 30 个采样周期的学习调节时间。在传统的信号处理应用中，信号的采样频率可取为 20kHz。而考虑在这样的采样频率下，30 个采样周期将意味着只要花 1.5 微秒的学习调节时间。

例如，这种自适应学习的线性网络可以用于监控系统中，当它跟踪的信号可能引起不稳定时，系统就会发出报警信号。

自适应学习线性网络模型的另一个应用则基于其能够对一个非线性系统进行逼近，并找到稳定的解，使得结果的误差平方和最小。当非线性系统在一个给定的点上稳定地进行运行的时候，自适应学习线性网络模型能够非常精确地对它进行逼近。如果非线性系统跳到另一个点上运行，则自适应学习线性网络模型的工作点也能够随之改变，并且跟踪到相应的新的工作点。



采样速率必须足够快，使得自适应线性网络模型能够在最短的时间里保持与非线性系统在当前工作点上是一致的。但是，自适应线性网络模型还是要花一定的时间，用以得到关于非线性系统的足够多的信息，从而实现自适应的学习。为了将这个学习时间尽量减少，可以在非线性系统的输入信号上加上少量的噪声。这样做使得非线性系统在一个很短的时间里能够动态地增加更多的运行点，从而使得自适应线性网络能够更学习得更快。当然，噪声要足够地小，使得其被加上以后不会影响系统的正常使用。

4.5 线性系统辨识

线性神经网络能够用于对实际系统的建模。当实际系统是一个线性系统或者是接近线性系统时，线性神经网络能够具备零误差或很小的误差。

4.5.1 问题描述

对于一个线性系统，假设其对输入信号进行线性放大后输出，其线性变换关系为 $y=kx+b$。其中 x 为输入，k 为放大倍数，b 为平移量，y 为此系统的输出。

对于这样的一个线性系统，我们假设持续时间为 5 秒，每 25 毫秒采样 1 次。

```
time=0:0.025:5;
X=sin(sin(time).*time*10);
plot(time,X);
title('输入信号 T');
xlabel('时间');
ylabel('输入信号');
```

在图中绘出输入信号 X 的曲线，如图 4-16 所示。

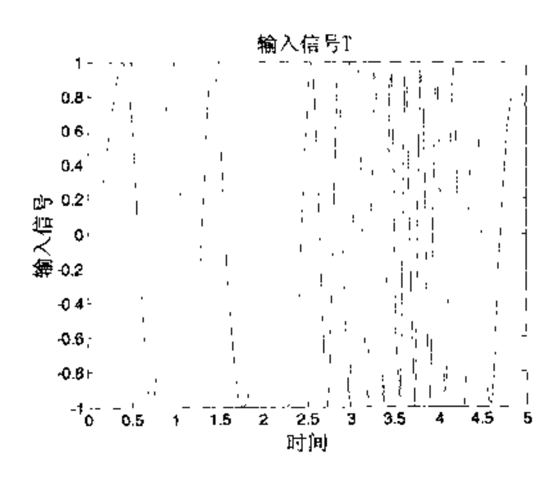


图 4-16　输入信号 X

由此线性系统的线性变换关系，我们可以得到其对输入信号 X 的输出 T 为（在这里我们选择 $k=2$，$b=0.8$）：

```
T=X*2+0.8;
plot(time,T);
title('系统输出 T');
xlabel('时间');
ylabel('系统输出');
```

在图中绘出系统输出信号 T 的曲线，如图 4-17 所示。

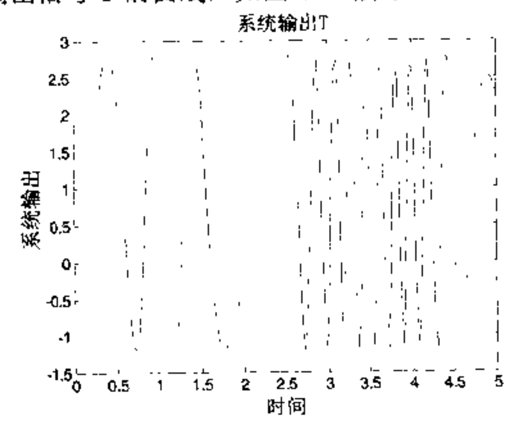


图 4-17　系统输出 T

网络的输入 P 为输入信号 X 的当前值，以及其前两个时刻的值，通过它们来预测系统的输出 T。

```
Q=size(X,2);
P=zeros(3,Q);
P(1,1:Q)=X(1,1:Q);
P(2,2:Q)=X(1,1:(Q-1));
P(3,3:Q)=X(1,1:(Q-2));
```

4.5.2 建立网络

由于系统只有一个输出，所以解决这个问题的网络只需要有一个神经元。这个神经元具有三个输入，其中一个为输入信号的当前值，另外两个为输入信号前两个时刻的值。

我们使用 newlind 函数来设计上述的神经元。

```
net=newlind(P,T);
```

4.5.3 测试网络

在设计好网络后，其权值和阈值就确定了。下面就可以测试这个网络的性能。我们使用函数 sim 仿真网络，计算出其对系统输出 T 的估计值 a。

```
a=sim(net,P);
```

为了比较系统输出和网络输出值之间的差别，我们把这两个信号在同一个图中绘出，如图 4-18 所示。

```
plot(time,a,'+',time,T,'--');
title('网络输出 a 与系统输出 T');
xlabel('时间');
ylabel('系统输出--  网络输出+');
```

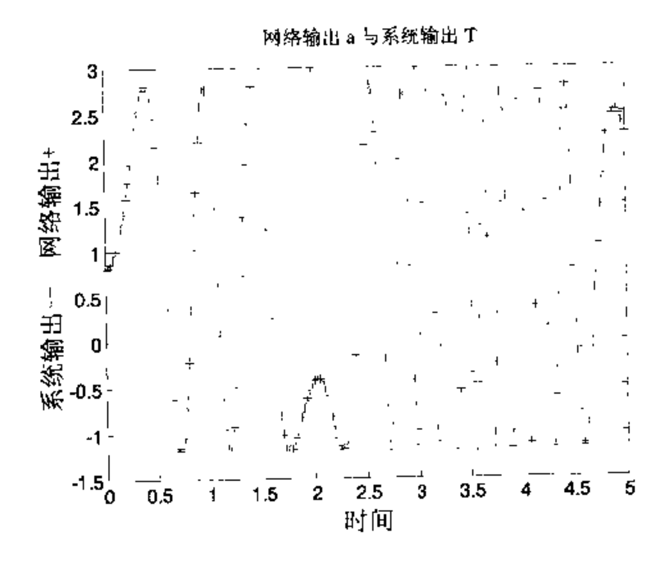


图 4-18 网络输出与系统输出

从图 4-18 中可以看出，系统输出信号和网络输出信号能够很好地吻合。这说明了我们设计的网络能够很好地工作。

为了比较出两种结果的细微差别，可求出网络输出的误差（即两种信号的差），绘出误差曲线，如图 4-19 所示。

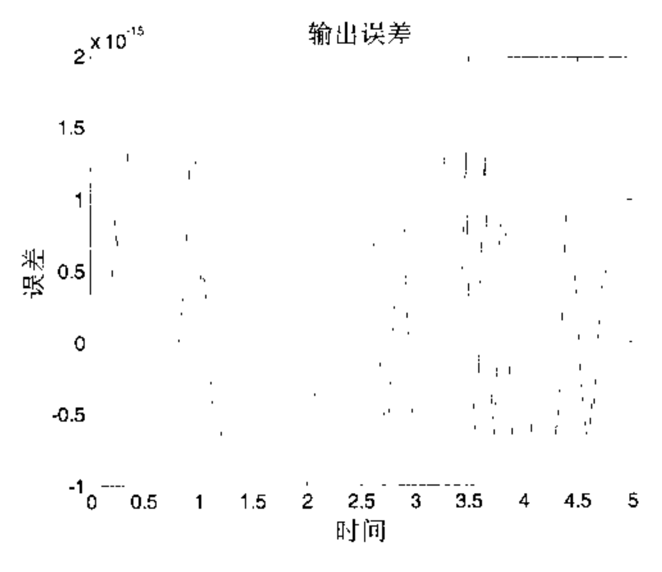


图 4-19　网络误差曲线

```
e=T-a;
figure;
plot(time,e);
title('输出误差');
xlabel('时间');
ylabel('误差');
```

可以看出，误差的量级为 10^{-15}，可见是非常小的。这正说明了网络的正确性。

例程 4-5 是本问题的 MATLAB 程序代码。

例程 4-5

```
%定义输入信号并绘出其曲线
time=0:0.025:5;
X=sin(sin(time).*time*10);
plot(time,X);
title('输入信号 T');
xlabel('时间');
ylabel('输入信号');
figure;
%定义系统线性变换函数,绘出系统输出曲线
T=X*2+0.8;
plot(time,T);
title('系统输出 T');
```

```
xlabel('时间');
ylabel('系统输出');
%定义网络输入
Q=size(X,2);
P=zeros(3,Q);
P(1,1:Q)=X(1,1:Q);
P(2,2:Q)=X(1,1:(Q-1));
P(3,3:Q)=X(1,1:(Q-2));
%建立网络
net=newlind(P,T);
%测试网络
a=sim(net,P);
%绘出网络输出 a 与系统输出 T
figure;
plot(time,a,'+',time,T,'--');
title('网络输出 a 与系统输出 T');
xlabel('时间');
ylabel('系统输出--   网络输出+');
%计算误差,并绘出其曲线
e=T-a;
figure;
plot(time,e);
title('输出误差');
xlabel('时间');
ylabel('误差');
```

4.5.4 结论

使用线性预测器，线性系统模型不仅能够用来对线性系统进行无误差的建模，也可以对非线性系统进行最小平方和误差意义下的建模。

当然，如果非线性系统的非线性程度太高，那么使用 BP 网络或者径向基网络更合适一些。

4.6 自适应系统辨识

下面的例子将设计一个线性网络来对一个线性系统进行自适应辨识。通过对网络的自适应训练，网络能够随着被辨识的模型的变化而相应地发生变化，从而实现对它的自适应辨识。

4.6.1 问题描述

给定一个信号 X，其持续时间为 6 秒，采样率为每秒钟 200 次。用这个信号对系统进行输入。

```
time1=0:0.005:4;
time2=4.005:0.005:6;
time=[time1 time2];
X=sin(sin(time*4).*time*8);
```

在图上绘出输入信号曲线，如图 4-20 所示。

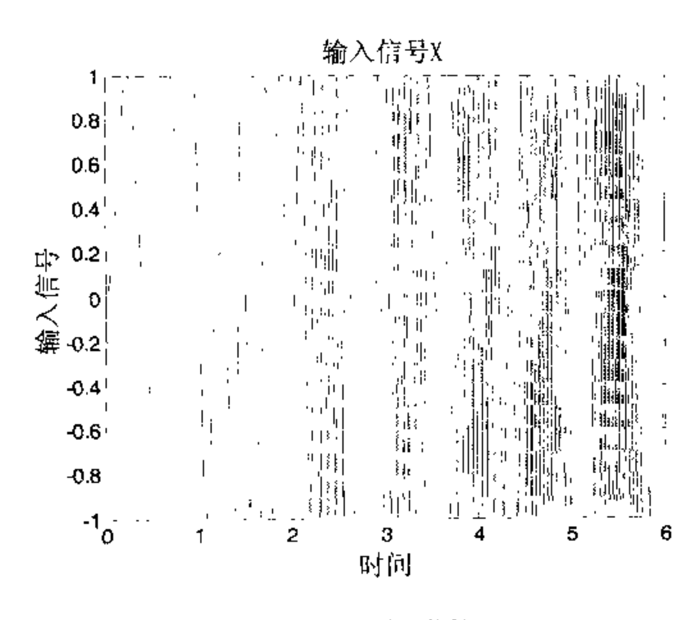


图 4-20　输入信号

```
plot(time,X);
title('输入信号 X');
xlabel('时间');
ylabel('输入信号');
```

系统的输出信号为 T，其定义如下所示。需要注意的是，系统对于 4 秒钟之前和之后又不同的响应。

```
steps1=length(time1);
[T1,state]=filter([1 -0.5],1,X(1:steps1));
steps2=length(time2);
T2=filter([0.9 -0.6],1,X((1:steps2) + steps1),state);
T=[T1 T2];
```

在图上绘出系统输出信号的曲线，如图 4-21 所示。需要注意的是，虽然输入信号在 4 秒前后有较大的变化，但经过这个较复杂的系统后，变化就不是很明显了。

```
plot(time,T);
title('系统输出 T');
xlabel('时间');
ylabel('系统输出');
```

网络的输入 P 为系统的输入信号 X。网络通过输入信号的最后两个时刻的值来估计系统的输出 T。我们需要将输入和输出信号转换成序列信号，这样才能保证网络能被不断地训练。

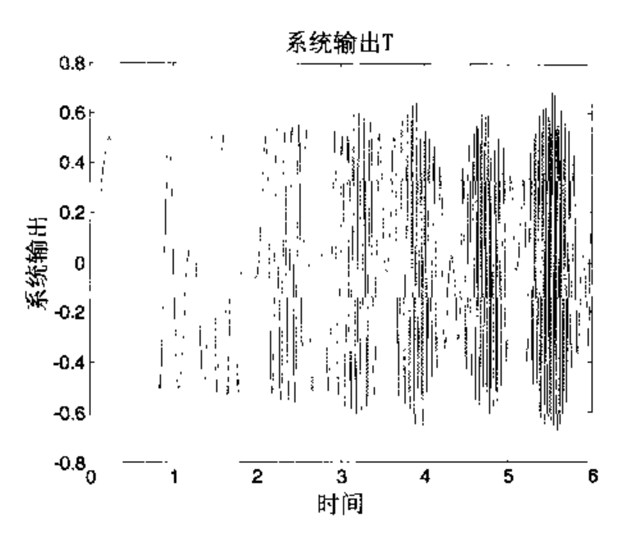


图 4-21　系统输出曲线

```
T=con2seq(T);
P=con2seq(X);
```

4.6.2　网络的建立

使用 newlin 函数来建立网络。这个函数能够为我们设计的具有两个输入的神经元提供权重和阈值。网络具有两个延迟输入，最后的两个输入信号将用于预测系统的输出。选择学习速率为 0.5，于是有：

```
lr=0.5;
delays=[0 1];
net=newlin(minmax(cat(2,P{:})),1,delays,lr);
```

4.6.3　网络训练

网络建立起来后，就要对它训练了。网络的权重和阈值现在就可以使用 adapt 函数训练得到。学习速率已经在 newlin 函数中被设置为 0.5。

```
[net,a,e]=adapt(net,P,T);
```

4.6.4　网络测试

网络经过训练后，权值和阈值都已经选择好了。为了测试网络的工作情况，我们将系统的实际输出 T 和网络的估计输出值 a 在同一个图中绘出，并对它们加上不同的标记，以便于观察，如图 4-22 所示。

```
plot(time,cat(2,a{:}),'+',time,cat(2,T{:}),'--');
title('网络输出 a 与系统输出 T');
xlabel('时间');
ylabel('系统输出--　网络输出+');
```

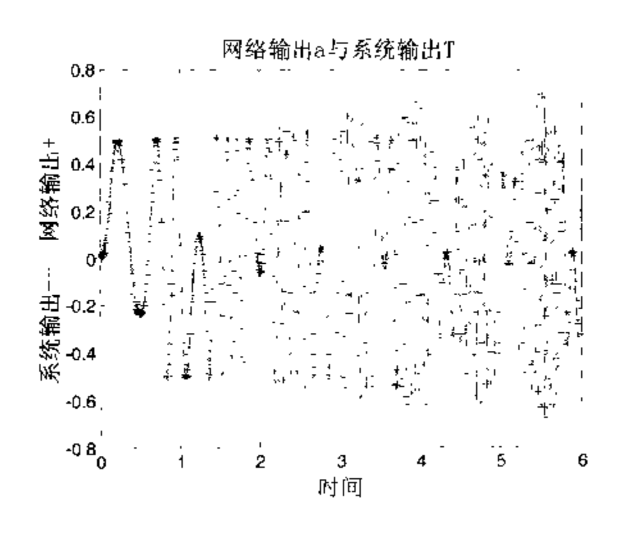


图 4-22　网络输出 a 与系统实际输出 T

由于信号过于复杂，在图 4-22 中两个信号并不能看得很清晰，于是我们可以绘出误差曲线来进行观察，如图 4-23 所示。

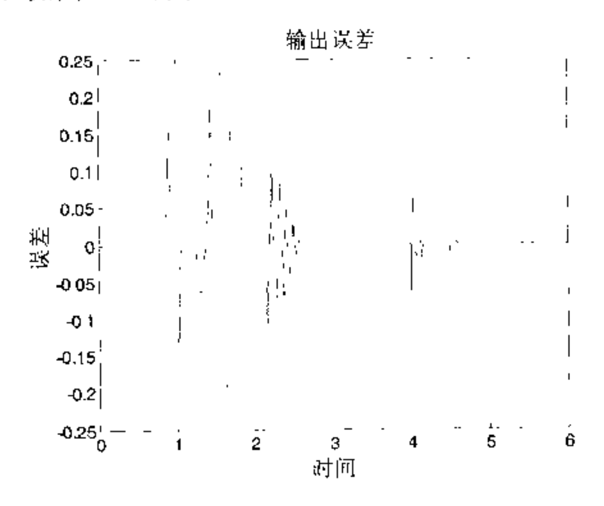


图 4-23　误差信号曲线

```
plot(time,cat(2,e{:}));
title('输出误差');
xlabel('时间');
ylabel('误差');
```

从图 4-23 中可以看出，网络花了 2.5 秒的时间来跟踪模型，以后就变得十分精确了。而到了第 4 秒时，系统发生了突变，网络又花了 0.2 秒的时间来再次跟踪模型。

例程 4-6 是本问题的 MATLAB 程序。

例程 4-6

```
%定义输入信号并绘出其曲线
```

```
time1=0:0.005:4;
time2=4.005:0.005:6;
time=[time1 time2];
X=sin(sin(time*4).*time*8);
plot(time,X);
title('输入信号 X');
xlabel('时间');
ylabel('输入信号');
%定义系统输出,绘出曲线
steps1=length(time1);
[T1,state]=filter([1 -0.5],1,X(1:steps1));
steps2=length(time2);
T2=filter([0.9 -0.6],1,X((1:steps2) + steps1),state);
T=[T1 T2];
figure;
plot(time,T);
title('系统输出 T');
xlabel('时间');
ylabel('系统输出');
%将系统输入和输出转换成序列信号
T=con2seq(T);
P=con2seq(X);
%建立网络
lr=0.5;
delays=[0 1];
net=newlin(minmax(cat(2,P{:})),1,delays,lr);
%训练网络
[net,a,e]=adapt(net,P,T);
%绘出网络输出 a 与系统输出 T
figure;
plot(time,cat(2,a{:}),'+',time,cat(2,T{:}),'--');
title('网络输出 a 与系统输出 T');
xlabel('时间');
ylabel('系统输出--   网络输出+');
%绘出误差曲线
figure;
plot(time,cat(2,e{:}));
title('输出误差');
xlabel('时间');
ylabel('误差');
```

4.6.5 结论

系统的自适应线性神经网络模型可以被用于保持大量的信息。例如，自适应模型能够被用来在给定的时间里得到实际系统的特征。又如，自适应网络可以用于监控系统中，当

系统的值有可能引起不稳定时，就发出报警信号。

自适应线性模型的另一个用途是利用它能够找到一个线性估计，对非线性系统行为进行最小平方和误差估计。

当非线性系统在一个给定的运行点工作时，自适应线性模型能够以高精度对它进行逼近。而一旦非线性系统转到了另一个运行点，自适应线性网络也会相应地转到那个新的点上工作。

为了保证线性系统模型对非线性系统在最短的时间跟踪到当前运行点，采样速率要足够地快。然而，为了得到非线性系统足够多的信息，以对其进行合理的建模，网络需要有一个最小的认知时间。为了尽量减小这个时间，可以对非线性系统的输入信号加入小量的噪声。这样做使得非线性系统在一个很短的时间里能够动态地增加更多的运行点，从而使得自适应线性网络能够学习得更快。当然，噪声要足够地小，使得其被加上以后不会影响系统的正常使用。

4.7　函数逼近

在人工神经网络的实际应用中，绝大部分的神经网络模型使用的是 BP 网络或其变化形式，它也是前向型神经网络的核心部分。

误差反传算法的主要思想是把学习过程分为两个阶段：第一阶段（正向传播过程），给出输入信息通过输入层经隐含层逐层处理并计算每个单元的实际输出值；第二阶段（反向过程），若在输出层未能得到期望的输出值，则逐层递归的计算实际输出与期望输出之差值（即误差），以便根据此差调节权值。

BP 网络有很强的映射能力，其主要用于：

- 模式识别、分类。用于语言、文字、图像的识别，用于医学特征的分类、诊断等。
- 函数逼近。用于非线性控制的函数的建模、机器人的轨迹控制及其他工业控制等。
- 数据压缩。在通信中的编码压缩和恢复，图像数据的压缩和存储，以及图像特征的抽取等。

本节介绍 BP 网络在函数逼近方面的应用，其他方面在后面会有介绍。

下面先设计一个简单的 BP 网络，用于对非线性函数的逼近。通过改变该函数的参数以及 BP 网络隐层神经元的数目，来观察训练时间及误差的变化情况。

4.7.1　问题描述

设我们要逼近的非线性函数为正弦函数，其频率参数可以调节。

```
p=[-1:.05:1];
t=sin(k*pi*p);
plot(p,t, '-')
title('要逼近的非线性函数');
xlabel('时间');
ylabel('非线性函数');
```

设 k=1，绘制出其曲线，如图 4-24 所示。

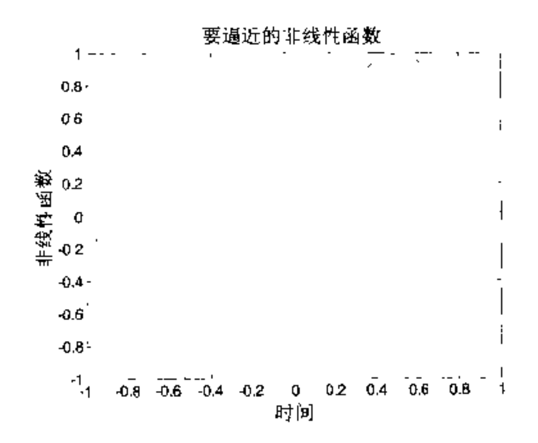


图 4-24 要逼近的非线性函数曲线

4.7.2 网络建立

使用函数 newff 建立 BP 网络，其中隐单元的神经元数目 n 可以改变，我们在这里先取其数目为 10。另外，选择各神经单元的传递函数分别为 tansig 函数和 purelin 函数，设置 BP 网络反传函数为 trainlm。

```
net=newff(minmax(p),[n,1],{'tansig' 'purelin'},'trainlm');
```

在对建好的网络进行训练之前，先看看仿真的效果。

```
y1=sim(net,p);
```

对仿真得到的结果绘出其曲线，并与原函数进行比较，如图 4-25 所示。

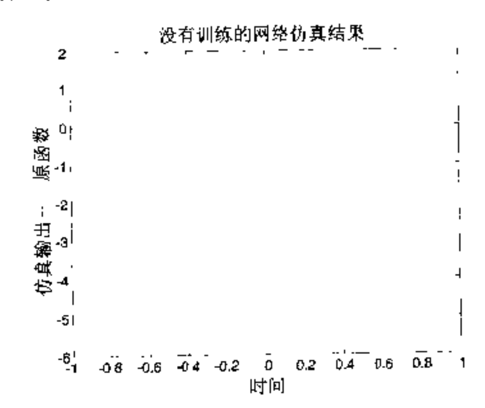


图 4-25 未经训练的网络输出与原函数比较

```
plot(p,t,'-',p,y1,'--')
title('没有训练的网络仿真结果');
xlabel('时间');
ylabel('仿真输出-- 原函数-');
```

由于使用 newff 建立网络时，对权值和阈值进行初始化是随机的，所以未经训练的网

络输出效果很差，而且每次运行时结果也不相同。

4.7.3 网络训练

下面进行训练。使用函数 train 对网络进行训练之前，必须先设置训练参数。我们设置训练时间为 50 个单位时间，训练目标为误差小于 0.01，其他参数使用默认值。得到的训练过程误差变化如图 4-26 所示。

```
net.trainParam.epochs = 50;
net.trainParam.goal = 0.01;
net = train(net,p,t);
```

在利用 train 函数对网络进行训练执行以上代码后，在 MATLAB 命令行中将实时地显示出网络的训练状态。如下所示：

```
TRAINLM, Epoch 0/50, MSE 14.2629/0.01, Gradient 325.276/1e-010
TRAINLM,Epoch2/50,MSE0.00126849/0.01,Gradient0.47402/1e-010
```

从图 4-26 中可以看出，对此网络的训练过程非常快，在经过两个单位时间以后，网络的误差就达到了要求。

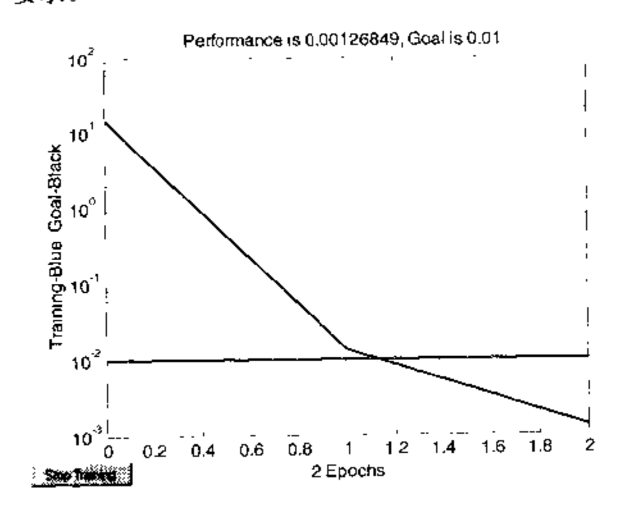


图 4-26　训练过程误差曲线

4.7.4 网络测试

对训练好以后的网络进行仿真。

```
y2 = sim(net,p);
```

仿真得到的结果如图 4-27 所示。同时将原始非线性函数曲线和未经训练的网络仿真结果都在此图中绘出。

```
plot(p,t,'-',p,y1,'--',p,y2,'--')
title('训练后的网络仿真结果');
xlabel('时间');
ylabel('仿真输出');
```

从图 4-27 中可以看出，这次得到的曲线与原始曲线很接近。这说明经过训练以后，网络对非线性函数的逼近效果相当好。

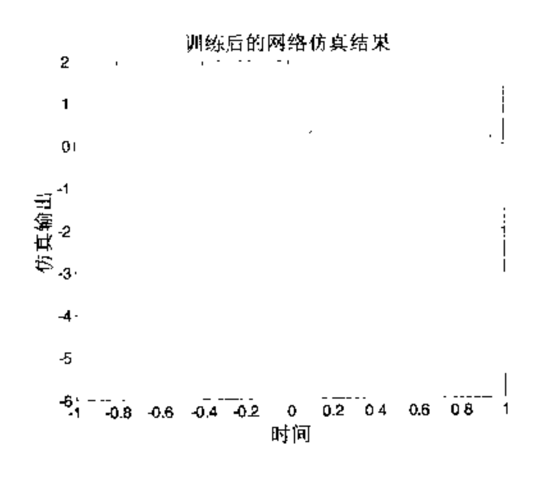


图 4-27　训练后的网络输出及比较

4.7.5 讨论

可以试着改变原始非线性函数的频率及 BP 网络隐层单元神经元的数目，重复上面的过程，观察它们对网络性能的影响。由于篇幅关系，本处就不赘述了。可以断定，非线性函数的频率越高，则对网络的要求也越高。相应地，网络隐层单元神经元数目越多，也就越能对非线性程度越高的函数进行更好地逼近。

例程 4-7 是本问题的 MATLAB 程序代码。

例程 4-7

```
%通过下面两个参数值的改变比较结果
k=1; %设置非线性函数的频率
n=10; %设置网络隐单元的神经元数目
%定义要逼近的非线性函数
p = [-1:.05:1];
t = sin(k*pi*p);
plot(p,t,'-')
title('要逼近的非线性函数');
xlabel('时间');
ylabel('非线性函数');
%建立相应的 BP 网络
net = newff(minmax(p),[n,1],{'tansig' 'purelin'},'trainlm');
%对没有训练的网络进行仿真
y1 = sim(net,p);
%绘出仿真得到的曲线
figure;
plot(p,t,'-',p,y1,'--')
```

```
title('没有训练的网络仿真结果');
xlabel('时间');
ylabel('仿真输出-- 原函数-');
%训练网络
net.trainParam.epochs = 50;
net.trainParam.goal = 0.01;
net = train(net,p,t);
%对训练后的网络进行仿真
y2 = sim(net,p);
%绘出训练后的仿真结果
figure;
plot(p,t,'-',p,y1,'--',p,y2,'--')
title('训练后的网络仿真结果');
xlabel('时间');
ylabel('仿真输出');
```

4.8 胆固醇含量估计

前面利用一个简单的非线性函数逼近的例子介绍了使用 BP 网络的方法，下面介绍一个实际中使用 BP 网络的例子。

4.8.1 问题描述

这是一个将神经网络用于医疗应用的例子。我们将设计一个器械，用于从血样的光谱组成的测量中得到血清的胆固醇含量级别。我们有 264 个病人的血样值，包括 21 种波长的谱线的数据。对于这些病人，我们同样得到了基于光谱分类的胆固醇含量级别 hdl，ldl，vldl。第一步，先将这些数据读入工作空间中，对这些数据进行规范化处理，然后对它们进行主要成分的分析。在 MATLAB 6.1 中有一个文件 choles_all.mat，其中包含了我们在本问题中需要的数据。

```
load choles_all
[pn,meanp,stdp,tn,meant,stdt]=prestd(p,t);
[ptrans,transMat]=prepca(pn,0.001);
```

在这里，我们删除了一些数据，适当地保留了变化不小于 0.01 的数据。下面来看转换后的数据矩阵的大小。

```
[R,Q]=size(ptrans)
```

得到的结果为：

```
R =
4
Q =
264
```

可以看出，通过主要成分分析，我们将输入向量从 21 个减少到了 4 个，可见原始数据有很大的冗余。

下一步将这些数据分成几个部分，分别用于训练、确证和测试。将数据的四分之一用于确证，四分之一用于测试，剩下的二分之一用于训练网络。等间隔地在原始数据中抽取出这些数据来。

```
iitst=2:4:Q;
iival=4:4:Q;
iitr=[1:4:Q 3:4:Q];
val.P=ptrans(:,iival);
val.T=tn(:,iival);
test.P=ptrans(:,iitst);
test.T=tn(:,iitst);
ptr=ptrans(:,iitr);
ttr=tn(:,iitr);
```

4.8.2 建立网络

现在开始建立网络。在本问题中，使用一个两层的网络。在隐含层中使用传递函数 tan-sigmoid，在输出层中使用一个线性传递函数。在函数逼近问题或者说是回归问题中，这种结构的 BP 网络是非常有用的。初步地，在隐含层中设计 5 个神经元单元（当然，这是含有很大的猜测成分的）。由于需要得到的是 3 个目标，所以网络需要有 3 个输出。

```
net=newff(minmax(ptr),[5 3],{'tansig' 'purelin'},'trainlm');
```

4.8.3 网络训练

使用 Levenberg-Marquardt 算法来训练网络。

```
[net,tr]=train(net,ptr,ttr,[],[],val,test);
```

在利用 train 函数对网络进行训练执行以上代码后，在 MATLAB 命令行中将实时地显示出网络的训练状态。如下所示：

```
TRAINLM, Epoch 0/100, MSE 3.90837/0, Gradient 847.755/1e-010
TRAINLM, Epoch 5/100, MSE 0.384139/0, Gradient 26.2029/1e-010
TRAINLM, Epoch 10/100, MSE 0.34918/0, Gradient 46.0177/1e-010
TRAINLM, Epoch 12/100, MSE 0.345209/0, Gradient 33.7444/1e-010
TRAINLM, Validation stop.
```

在 12 个训练单位以后，由于确证误差的增加，训练停止。分别将训练误差、确证误差和测试误差曲线在同一幅图中绘出，这样可以更加直观地观察到训练的过程。使用下列的代码得到训练过程误差，如图 4-28 所示。

```
plot(tr.epoch,tr.perf,'r',tr.epoch,tr.vperf,':g',
tr.epoch,tr.tperf,'-.b');
legend('训练','确证','测试',-1);
ylabel('平方误差');
xlabel('时间');
```

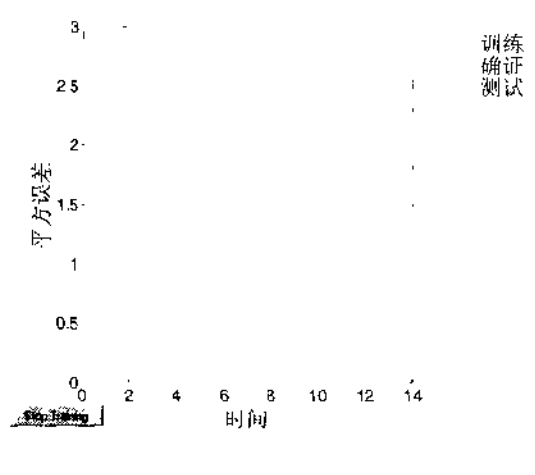


图 4-28　训练过程

上面得到的结果是合理的，测试集合误差和确证集合的误差有相似的性质。

4.8.4　分析及讨论

下面对网络的响应进行一些分析。我们将所有的数据（包括训练、确证和测试）通过网络，然后对网络输出和相应的目标进行线性回归。在这之前，要对网络的输出进行反规范化变换。

```
an=sim(net,ptrans);
a=poststd(an,meant,stdt);
for i=1:3
figure(i)
[m(i),b(i),r(i)]=postreg(a(i,:),t(i,:));
end
```

这样，得到了三组输出，所以进行三次线性回归。结果分别如图 4-29（a）、4-29（b）、4-29（c）所示。

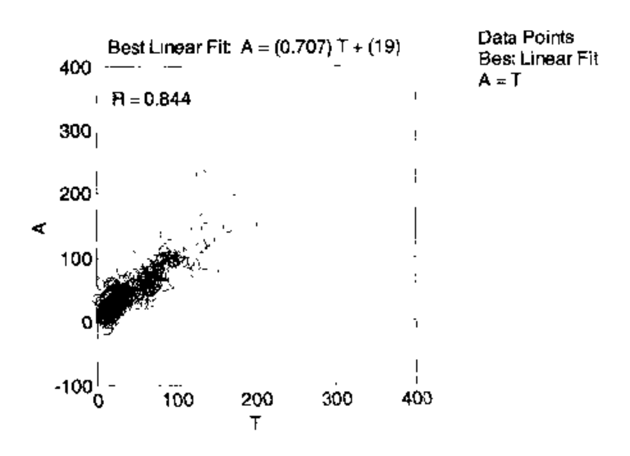


图 4-29（a）　级别为 hdl 的线性回归

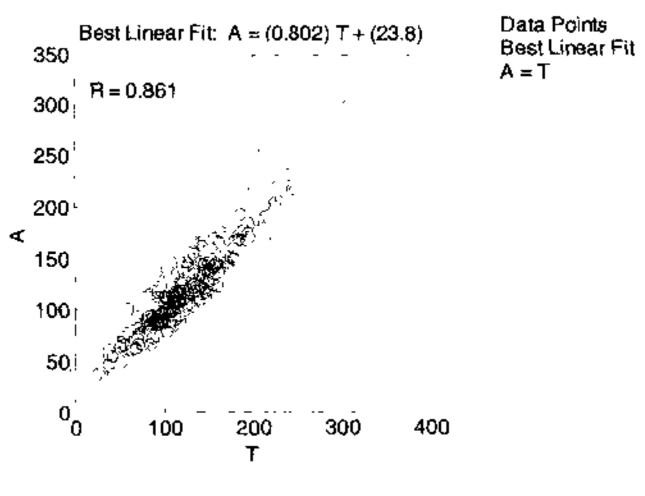


图 4.29（b） 级别为 ldl 的线性回归

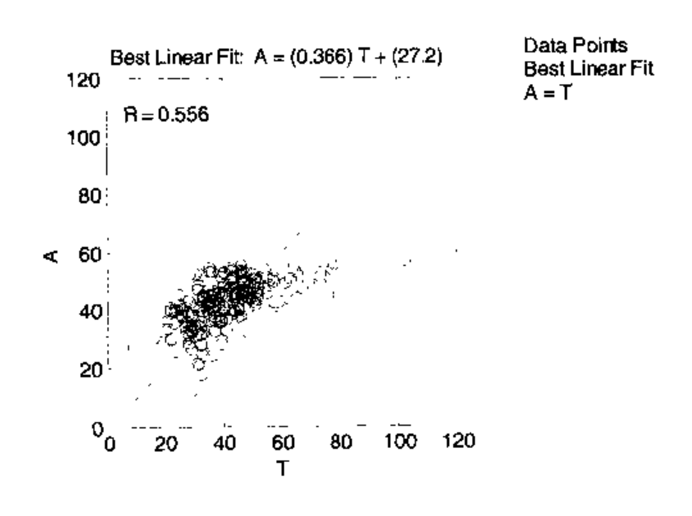


图 4-29（c） 级别为 vldl 的线性回归

可以看出，前面的两个输出对目标的跟踪比较好（毕竟，这是个很复杂的问题），相应的 *R* 值接近 0.9。而第三输出（vldl 级别）却并不很理想。我们很可能需要在这一点上做更多的工作。可能需要使用其他的网络结构（使用更多的隐含层神经元），或者是在训练技术上使用贝叶斯规范化方法而不是使用早停的方法。当然， vldl 级别也有可能本来就不能用给定的光谱组成来进行精确的计算。

有兴趣的读者可以自己试着对这个网络模型进行改进，以得到更好的效果。这里再给出将隐层神经元数目改为 20 时得到的结果。与前面的图 4-28、图 4-29（a）、图 4-29（b）、图 4-29（c）对应，分别得到图 4-30、图 4-31（a）、图 4-31（b）、 图 4-31（c）。

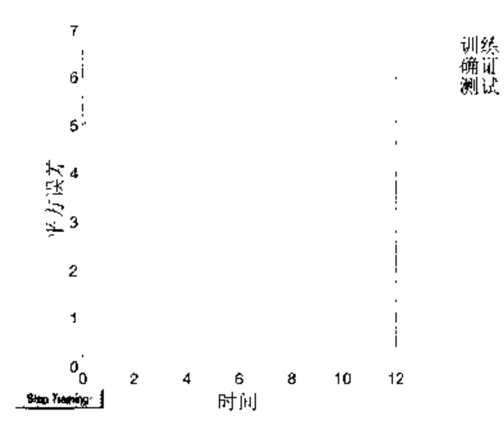


图 4-30　训练过程

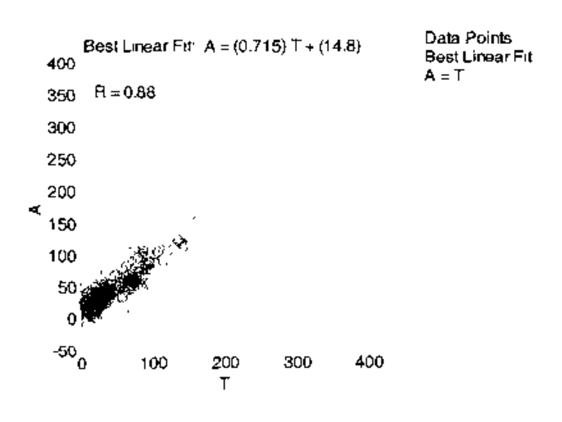


图 4-31（a）　级别为 hdl 的线性回归

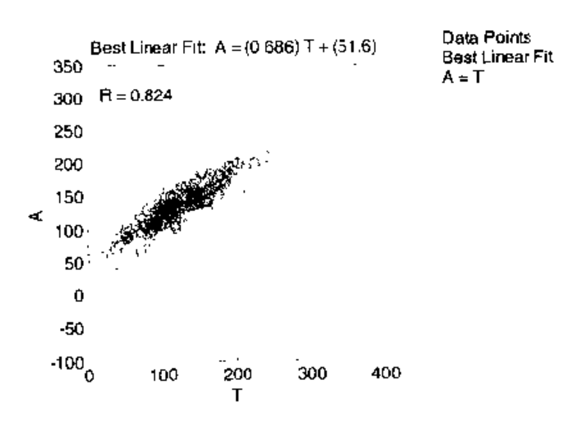


图 4-31（b）　级别为 ldl 的线性回归

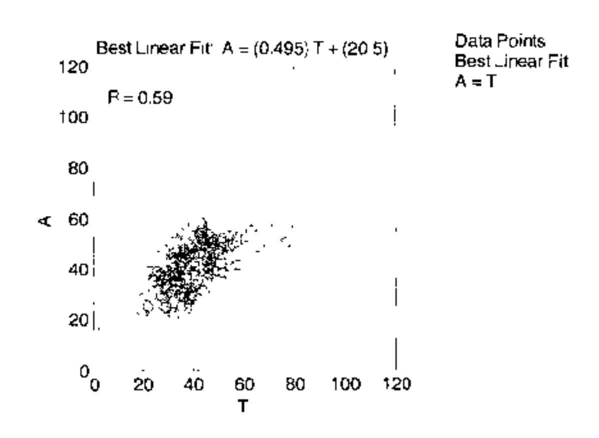


图 4-31（c） 级别为 vhdl 的线性回归

可以看出，隐层神经元单元的数目增加为 20 以后，网络训练过程的三种误差非常接近，而相应的对目标结果和输出结果进行线性回归，得到的结果为：前面两种回归效果较好，但比神经单元增加前反而要差一点，第三种回归效果则比增加前要好一些。相应的分析在这里就不再给出了，有兴趣的读者可以自己试着给出。

例程 4-8 是本问题的 MATLAB 程序代码。

例程 4-8

```
%导入原始测量数据
load choles_all;
%对原始数据进行规范化处理
[pn,meanp,stdp,tn,meant,stdt]=prestd(p,t);
[ptrans,transMat]=prepca(pn,0.001);
[R,Q]=size(ptrans)
%将原始数据分成几个部分作为不同用途
iitst=2:4:Q;
iival=4:4:Q;
iitr=[1:4:Q 3:4:Q];
vv.P=ptrans(:,iival);
vv.T=tn(:,iival);
vt.P=ptrans(:,iitst);
vt.T=tn(:,iitst);
ptr=ptrans(:,iitr);
ttr=tn(:,iitr);
%建立网络
net=newff(minmax(ptr),[5 3],{'tansig' 'purelin'},'trainlm');
%训练网络
net.trainParam.show=5;
[net,tr]=train(net,ptr,ttr,[],[],vv,vt);
%绘出训练过程中各误差的变化曲线
```

```
plot(tr.epoch,tr.perf,'r',tr.epoch,tr.vperf,':g',
tr.epoch,tr.tperf,'-.b');
legend('训练','确证','测试',-1);
ylabel('平方误差');
xlabel('时间');
pause;
%得到各个级别的结果的线性回归结果，并绘出曲线
an=sim(net,ptrans);
a=poststd(an,meant,stdt);
for i=1:3
figure(i)
[m(i),b(i),r(i)] = postreg(a(i,:),t(i,:));
end
```

4.8.5 结论

多层网络能够对任意的线性或者非线性函数进行逼近，其精度也是任意的。这样的网络有感知器网络和线性网络无法比拟的优势。但是，需要注意的是，存在这样的情况，虽然在训练时网络表现出很好的性能，但是 BP 网络并不一定总能找到解。

另外，虽然在上面没有提到，但是在训练时，学习速率和训练算法对网络的性能都是有影响的。学习速率太快可能引起不稳定，相反，学习速率太慢则要花费太多的时间。而且，不像线性网络那样，对于非线性多层网络，BP 网络没有很好的办法刚好找到的合适的学习速率。

最后，BP 网络对隐含层的神经元数目也是很敏感的。神经元太少，网络就很难适应，而太多的话，又可能设计出超适应的网络。

4.9 特 征 识 别

使用机器来进行模式的识别是一项非常有用的工作。能够辨别符号的机器是很有价值的。如果机器能识别银行的签字，那么它就能在相同的时间里做比人多得多的工作。这种引用不仅省时，而且省钱，同时还能将那个做重复工作（辨别签字）的人解放出来。下面就用 BP 网络的方法来设计一个能完成特征识别任务的网络。

4.9.1 问题描述

设计一个网络并训练它来识别字母表中的 26 个字母。这些字母已经被数字成像系统数字化了。其结果是对应每个字母有一个 5×7 的布尔量网格。例如，图 4-32 中显示的就是字母 A 的网格图。

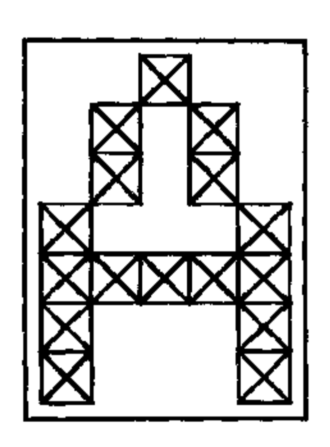


图 4-32　字母 A 的网格图

但是，图 4-32 只是理想图像系统得到的结果，实际中的图像系统总会存在一些噪声干扰或者是非线性因素，实际得到的字母的网格图如图 4-33 所示。

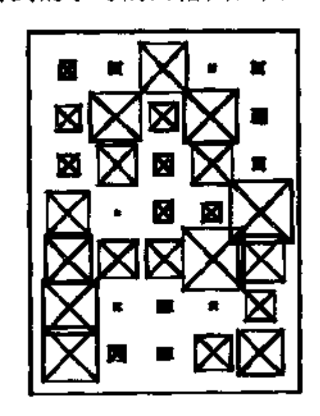


图 4-33　带有噪声的字母 A 的网格图

设计的网络要不仅能够对理想的输入向量进行很好的分类，也要对含有误差的输入向量有合理的准确度。

在本问题中，26 个含 35 个元素的输入向量被定义成一个输入向量矩阵 alphabet。目标向量也被定义成一个变量 targets。每个目标向量含有 26 个元素。向量代表某个字母，则其对应位置的元素值为 1，而其他位置的元素值为 0。例如，字母 A 对应的向量，其第一个位置的元素值为 1（因为 A 是字母表中的第一个字母），而从第 2~26 个位置的元素值均为 0。

4.9.2 神经元网络

设计的网络把 35 个布尔值作为一个具有 35 个元素的输入向量。需要网络通过输出一个具有 26 个元素的输出向量来区分字母。这个 26 元素向量的每一个代表着一个字母。在正常运行的情况下，对于一个输入字母，网络要能输出一个向量，它的对应位置元素值为

1，其他的值为 0。

除此之外，网络还必须能够有容错能力。在实际中，网络不可能接收到一个理想的布尔向量作为输入。假设设计的网络能够有一定的容错能力，对于输入向量，若其噪声均值为 0，标准差不大于 0.2，则能够分辨出来。

1. 网络结构

为了辨别字母，所设计的网络需要有 35 个输入，在输出层，则需要有 26 个神经元。我们设计一个有两层结构的 log-sigmoid/log-sigmoid 网络。之所以选择 log-sigmoid 函数，是因为它的输出范围（0 到 1）正好适合在学习后输出布尔值。

其网络结构如图 4-34 所示。

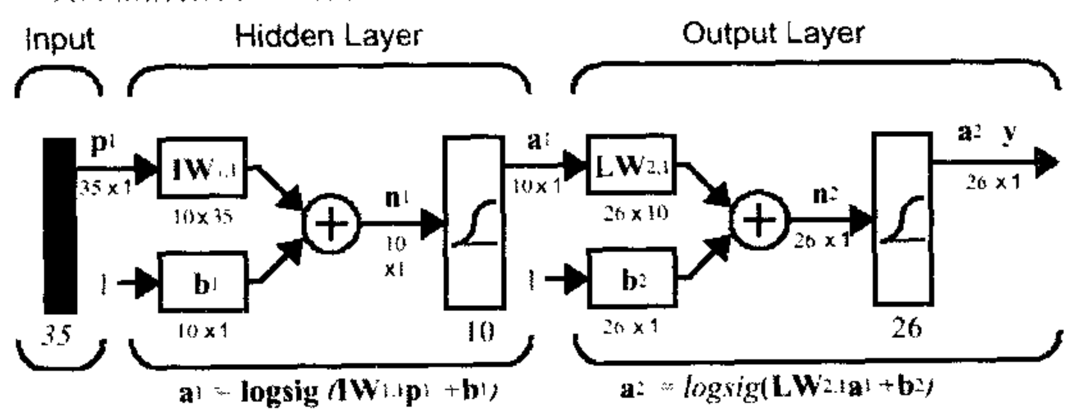


图 4-34　网络结构

在网络的隐含层（第一层）设计了 10 个神经元。关于神经元的数目选择不仅需要经验也还要有猜测的成分。

训练网络就是要使其将输出向量中正确的位置设置为 1，其余位置全为 0。然而，由于噪声信号的引入，网络就可能不会输出正确的 1 或 0 信号。在网络被训练后，将带有噪声的字母信号输入网络，就会在正确的位置上得到 1 和 0。

2. 初始化

使用函数 newff 创建一个两层网络。

```
S1=10;
[R,Q]=size(alphabet);% alphabet 在 prprob 函数中有定义
[S2,Q]=size(targets); % targets 在 prprob 函数中有定义
P=alphabet;
net=newff(minmax(P),[S1 S2],{'logsig' 'logsig'},'traingdx');
net.LW{2,1}=net.LW{2,1}*0.01;
net.b{2}=net.b{2}*0.01;
```

3. 网络训练

为了使产生的网络对输入向量有一定的容错能力，最好的办法是既使用理想的信号和又使用带有噪声的信号对网络进行训练。我们的具体做法是先用理想的输入信号对网络进行训练，直到其平方和误差足够小。

接下来，使用 10 组理想信号和带有噪声的信号对网络进行训练。在输入带有误差的向量时，要输入两倍重复的无误差信号，这样做的目的是为了保证网络在分辨理想输入向量时的稳定性。

在网络进行了上述的训练以后，网络对无误差的信号可能也会采用对付带有噪声信号的办法。这样做就会付出较大的代价。因此，我们可以再一次训练网络。这一次就只使用理想的向量进行训练。这样就可以保证在输入端输入理想字母信号时，网络能够最好地对其做出反应。

以上所有的训练都是使用 BP 网络来实现的。网络学习的速率和冲量参数设置为自适应改变。使用函数 trainbpx 进行快速训练。

4. 无噪声的训练

开始时使用无噪声的信号对网络进行训练。当训练时间达到 5000 个时间单位或者是网络平方和误差小于 0.1 时停止网络的训练。

```
P=alphabet;
T=targets;
net.performFcn='sse';
net.trainParam.goal=0.1;
net.trainParam.show=20;
net.trainParam.epochs=5000;
net.trainParam.mc=0.95;
[net,tr]=train(net,P,T);
```

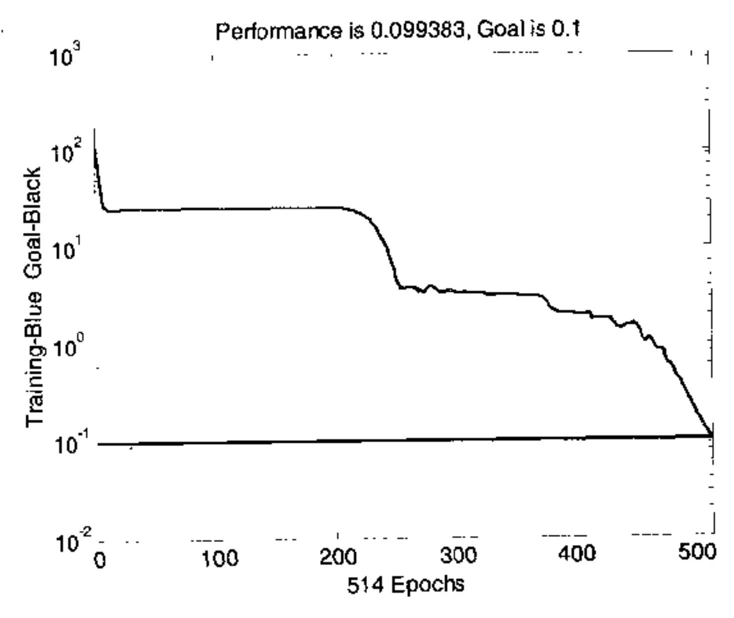


图 4-35 无噪声训练过程误差变化情况

训练过程误差变化情况如图 4-35 所示。观察其曲线可知训练指标能在较快的时间内达到。下面列出了在 MATLAB 命令行中实时地显示出的网络的训练状态。从这里也可以观察出误差的变化情况。

```
TRAINGDX,Epoch0/5000, SSE 168.123/0.1, Gradient 64.9982/1e-006
TRAINGDX,Epoch20/5000,SSE25.4428/0.1,Gradient0.43545/1e-006
TRAINGDX,Epoch40/5000,SSE25.8033/0.1,Gradient0.19349/1e-006
TRAINGDX,Epoch60/5000,SSE25.8669/0.1,Gradient0.15222/1e-006
TRAINGDX,Epoch80/5000,SSE25.8783/0.1,Gradient0.14765/1e-006
TRAINGDX,Epoch100/5000,SSE25.8748/0.1,Gradient0.1559/1e-006
TRAINGDX,Epoch120/5000,SSE25.8535/0.1,Gradient0.1859/1e-006
TRAINGDX,Epoch140/5000,SSE25.7436/0.1,Gradient0.3159/1e-006
TRAINGDX,Epoch160/5000,SSE25.1517/0.1,Gradient0.5746/1e-006
TRAINGDX,Epoch180/5000,SSE23.1155/0.1,Gradient0.7075/1e-006
TRAINGDX,Epoch200/5000,SSE18.0836/0.1,Gradient0.7858/1e-006
TRAINGDX,Epoch220/5000,SSE9.79391/0.1,Gradient0.6748/1e-006
TRAINGDX,Epoch240/5000,SSE8.3323/0.1,Gradient1.75179/1e-006
TRAINGDX,Epoch260/5000,SSE7.64511/0.1,Gradient1.5499/1e-006
TRAINGDX,Epoch280/5000,SSE7.07863/0.1,Gradient0.2263/1e-006
TRAINGDX,Epoch300/5000,SSE6.57326/0.1,Gradient0.5750/1e-006
TRAINGDX,Epoch320/5000,SSE4.91553/0.1,Gradient0.4303/1e-006
TRAINGDX,Epoch340/5000,SSE3.21507/0.1,Gradient0.4119/1e-006
TRAINGDX,Epoch360/5000,SSE1.49677/0.1,Gradient2.4136/1e-006
TRAINGDX,Epoch380/5000,SSE1.18337/0.1,Gradient0.4605/1e-006
TRAINGDX,Epoch400/5000,SSE1.37881/0.1,Gradient1.1874/1e-006
TRAINGDX,Epoch420/5000,SSE0.5995/0.1,Gradient0.2431/1e-006
TRAINGDX,Epoch440/5000,SSE0.43953/0.1,Gradient0.135/1e-006
TRAINGDX,Epoch460/5000,SSE0.2819/0.1,Gradient0.08366/1e-006
TRAINGDX,Epoch480/5000,SSE0.1531/0.1,Gradient0.03703/1e-006
TRAINGDX,Epoch493/5000,SSE0.0976/0.1,Gradient0.02192/1e-006
```

5. 含有噪声信号的训练

为了保证设计的网络对噪声不敏感，可用理想的字母表向量和加了噪声的字母表向量分别训练网络。设置向字母表向量加入的噪声信号平均值分别为 0.1 和 0.2。这样就可以保证神经元网络学会在辨别带噪声信号的字母表向量时，也能对理想的字母表向量有正确的识别。

另外，设置网络对含噪声信号的向量进行训练的最大时间为 300 个单位时间，并且把误差参数也增加到 0.6。之所以要提高误差参数，是因为这次的训练向量（其中一些向量含有误差）增加了四倍。

```
netn=net;
netn.trainParam.goal=0.6;
netn.trainParam.epochs=300;
T=[targets targets targets targets];
for pass=1:10
P=[alphabet, alphabet, ...
(alphabet+randn(R,Q)*0.1), ...
(alphabet+randn(R,Q)*0.2)];
[netn,tr]=train(netn,P,T);
end
```

在输入端字母表向量上加入噪声信号后，网络的训练过程误差变化情况如图 4-36（a）所示。由于这次的训练次数很多（10 次），故将它们的曲线图缩小后给出来，从图 4-36（a）～图 4-36（j）上能看到它们的变化规律。观察这些曲线可知训练指标能在很快的时间内达到。在训练的同时，在 MATLAB 命令行中能够实时显示出网络的训练状态。由于训练次数太多，故此处略去，有兴趣的读者可以自己试着用本程序的代码来得到它们。从这些状态输出中也可以观察出误差的变化情况。

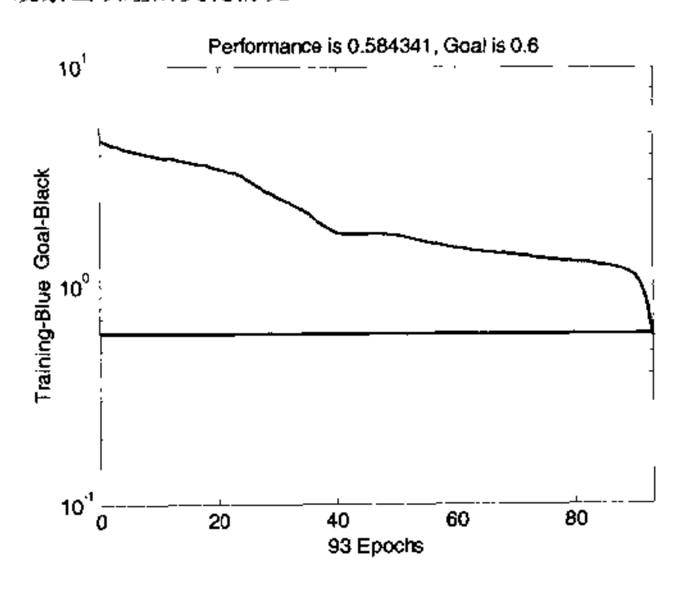


图 4-36（a） 有噪声训练过程误差变化情况

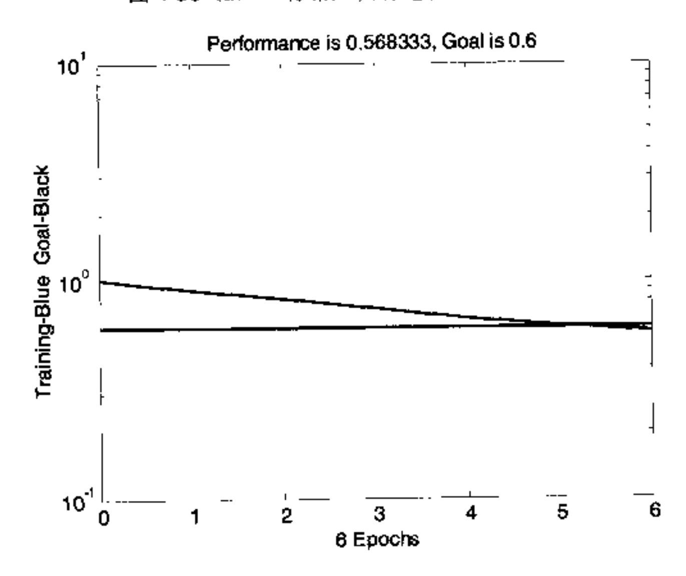


图 4-36（b） 有噪声训练过程误差变化情况

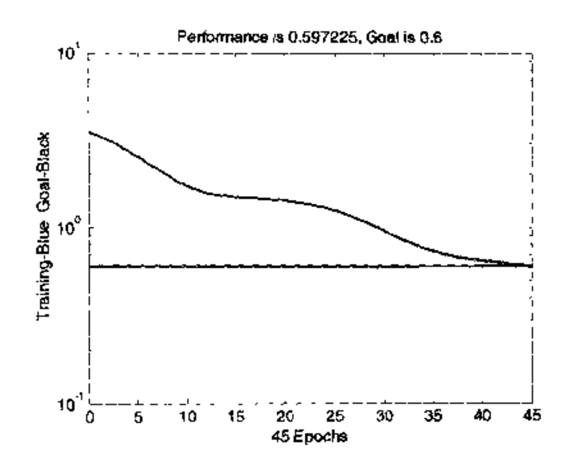


图 4-36（c）　有噪声训练过程误差变化情况

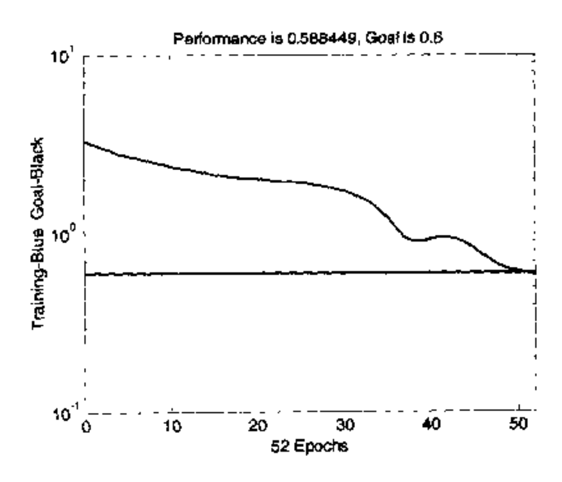


图 4-36（d）　有噪声训练过程误差变化情况

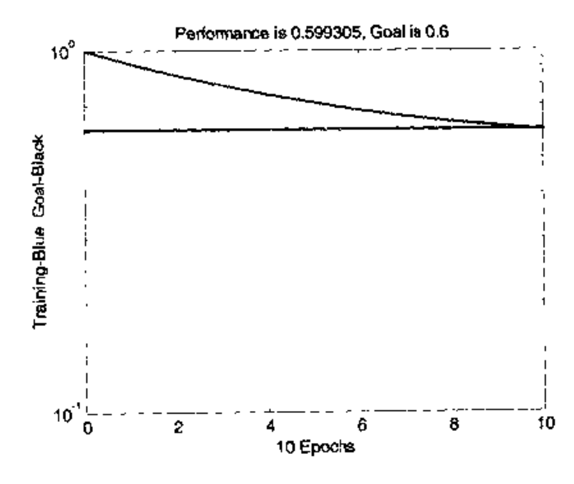


图 4-36（e）　有噪声训练过程误差变化情况

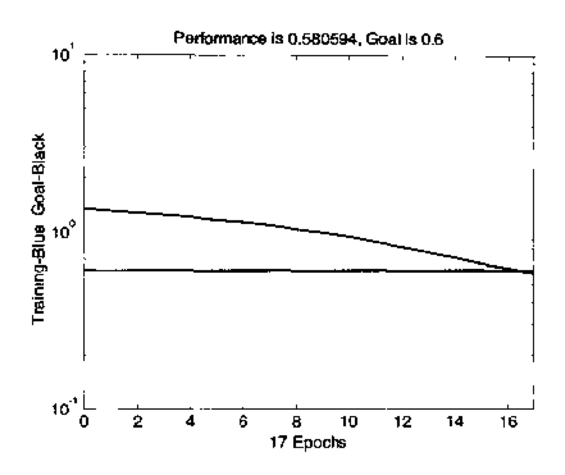


图 4-36（f） 有噪声训练过程误差变化情况

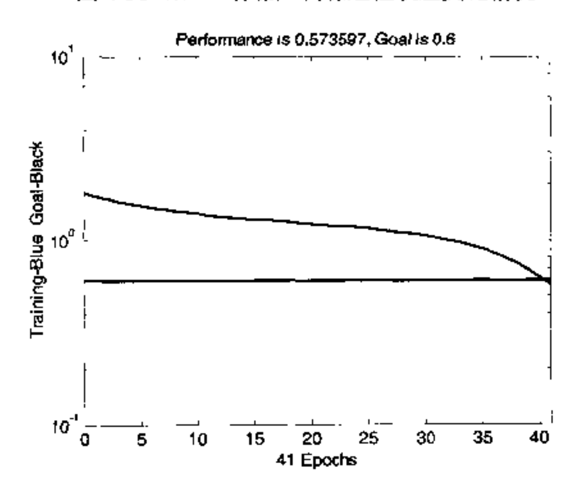


图 4-36（g） 有噪声训练过程误差变化情况

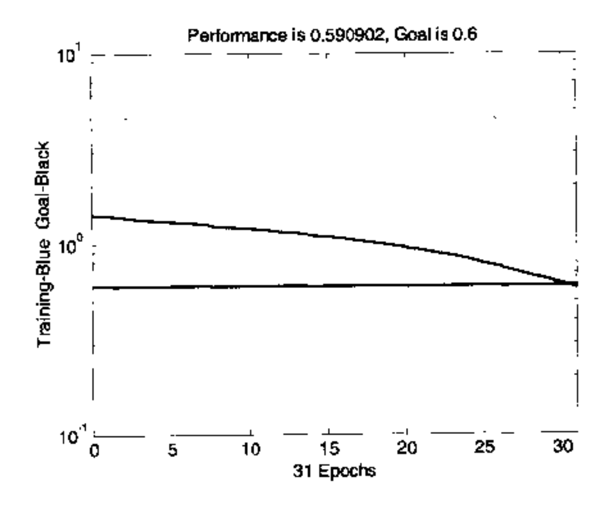


图 4-36（h） 有噪声训练过程误差变化情况

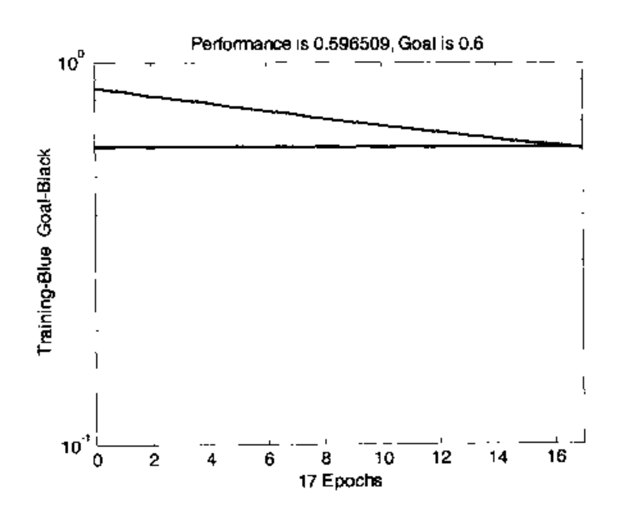


图 4-36（i）　有噪声训练过程误差变化情况

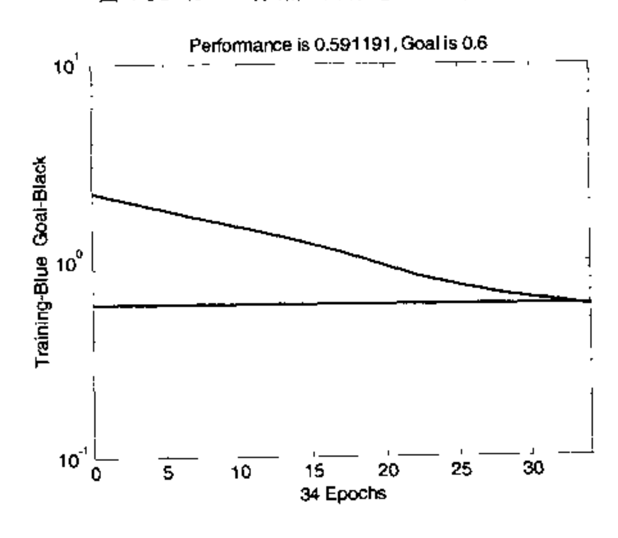


图 4-36（j）　有噪声训练过程误差变化情况

6. 再次对无噪声信号训练

上面已经用含噪声的信号对网络进行了训练，为了保证网络总是能够正确地对理想输入信号进行分类，我们需要再一次用无噪声的理想信号对网络进行训练。所以，使用与“含有噪声信号的训练”中相同的代码。具体代码从略。

4.9.3 系统性能评估

为了测试我们设计的神经网络模式识别系统的可靠性，我们用数百个加入了不同数量噪声的字母表向量作为输入，来观察其输出结果。

例程 4-9 是系统性能评估的代码。

例程 4-9

```
noise_range=0:0.05:0.5;
max_test=100;
T=targets;
for i=1:11
noiselevel(i)=noise_range(i);
errors1(i)=0;
errors2(i)=0;
for j=1:max_test
P=alphabet+randn(35,26)*noiselevel(i); % 测试未经误差训练的网络
A=sim(net,P);
AA=compet(A);
errors1(i)=errors1(i)+sum(sum(abs(AA-T)))/2; % 测试经过误差训练的网络
An=sim(netn,P);
AAn=compet(An);
errors2(i)=errors2(i)+sum(sum(abs(AAn-T)))/2;
end
end
pause
figure
plot(noise_range,errors1*100,'--',noise_range,errors2*100);
title('识别错误率');
xlabel('噪声指标');
ylabel('未经误差训练的网络 - -    经过误差训练的网络---');
```

在本问题中，使用不同级别的误差信号，并且绘出了网络输出错误与噪声信号的比较的曲线，如图 4-37 所示。

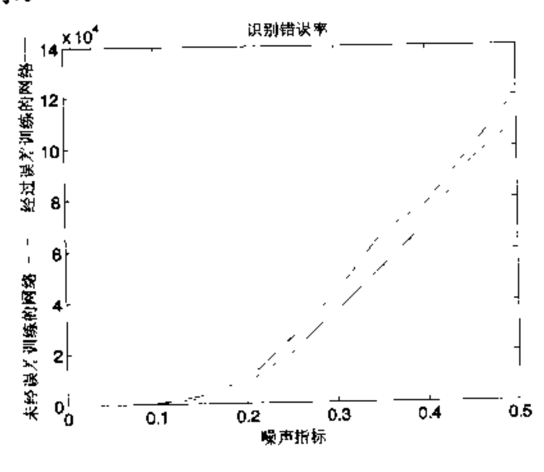


图 4-37　识别错误率曲线

加到网络输入向量上的噪声均值为 0，标准差范围为 0～0.5。在每个噪声级别上，分别有 100 个不同版本的噪声信号被加到每个字母向量上，然后用设计的网络计算其输出。

将输出通过竞争传递函数，保证 26 个输出（代表字母表中的字母）中只有一个的值为 1，其余均为 0。

图 4-37 中的实线显示的是既经过噪声信号又经过非噪声信号训练后的网络的可靠性。而虚线显示的则是同样的网络只经过非噪声信号而没有经过噪声信号训练的网络的可靠性。从图 4-37 上的曲线可以看出，网络经过含噪声信号的输入训练后，其容错能力有了非常明显的增强。

当输入向量的误差平均值为 0 或 0.05 时，网络识别没有错误。但是当误差平均值达到 0.2 时，两个网络都开始产生误差。

如果需要有更高的精度，一种办法是增长网络的训练时间，另一种办法是增加网络隐含层的神经元数目。当然，把输入的字母表向量从 5×7 的网格增加到 10×14 的网格也是一种办法。另外，如果网络要求对误差信号有更高的可靠性，还可以在训练时增加输入向量的误差的数量。

为了测试系统，可试验几个实际的字母 A、F、K、P、U、Z，并对它们加入误差信号，然后把它们输入到网络中，观察其得到的输出，如图 4-38（a）～图 4-38（d）所示。

```
for index=1:5:26
noisyJ=alphabet(:,index)+randn(35,1)*0.2;
figure;
plotchar(noisyJ);
A2=sim(net,noisyJ);
A2=compet(A2);
```

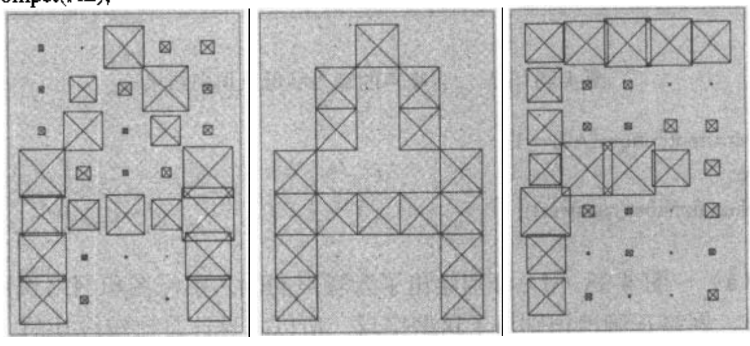


图 4-38（a）　含噪声的输入字母及识别结果

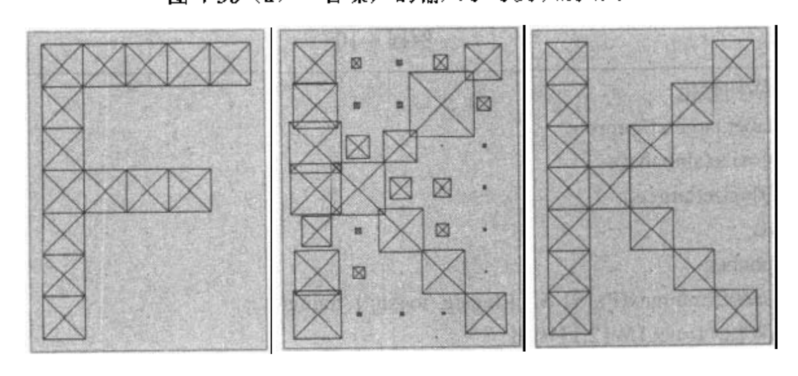


图 4-38（b）　含噪声的输入字母及识别结果一

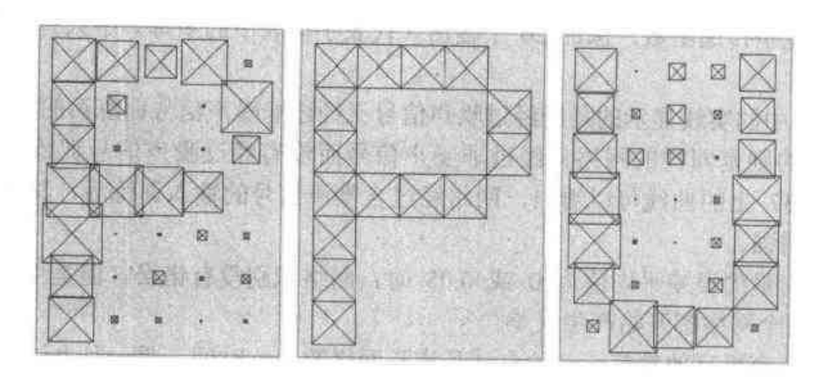


图 4-38（c） 含噪声的输入字母及识别结果二

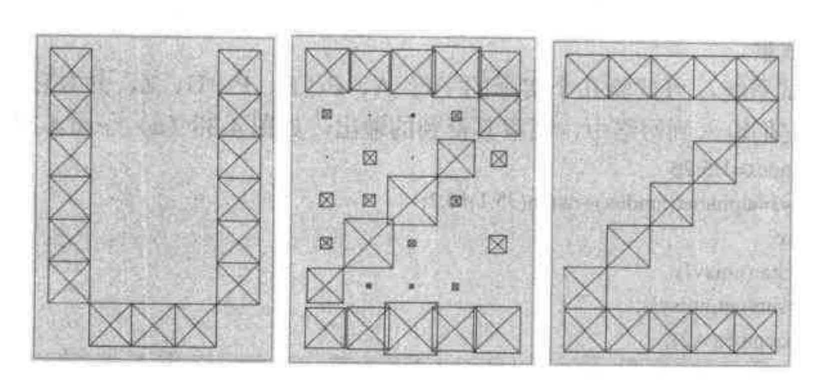


图 4-38（d） 含噪声的输入字母及识别结果二

```
answer=find(compet(A2)==1);
figure;
plotchar(alphabet(:,answer));
end
```

图 4-38（a）～图 4-38（d）分别绘出了含噪声的输入字母图和网络识别后的输出字母图。可以看出，网络正确地识别出了这些字母。所以本设计是比较成功的。

例程 4-10 是本问题的 MATLAB 程序代码。

例程 4-10

```
%网络初始化
[alphabet,targets]=prprob;
[R,Q]=size(alphabet);
[S2,Q]=size(targets);
S1=10;
P=alphabet;
net= newff(minmax(P),[S1 S2],{'logsig' 'logsig'},'traingdx');
net.LW{2,1}=net.LW{2,1}*0.01;
net.b{2}=net.b{2}*0.01;
```

```
T=targets;
%网络训练参数设置
net.performFcn='sse';
net.trainParam.goal=0.1;
net.trainParam.show=20;
net.trainParam.epochs=5000;
net.trainParam.mc=0.95;
%开始对无误差输入向量进行训练
[net,tr]=train(net,P,T);
%网络训练参数设置,并对有误差输入向量进行训练
netn=net;
netn.trainParam.goal=0.6;
netn.trainParam.epochs=300;
T=[targets targets targets targets];
pause
for pass=1:10
P=[alphabet,alphabet,...
(alphabet+randn(R,Q)*0.1),...
(alphabet+randn(R,Q)*0.2)];
[netn,tr]=train(netn,P,T);
pause
end
%网络再次对无误差输入向量进行训练
P=alphabet;
T=targets;
net.performFcn='sse';
net.trainParam.goal=0.1;
net.trainParam.show=20;
net.trainParam.epochs=5000;
net.trainParam.mc=0.95;
[net,tr]=train(net,P,T);
pause
%测试网络的容错性
noise_range=0:0.05:0.5;
max_test=100;
T=targets;
for i=1:11
noiselevel(i)=noise_range(i);
errors1(i)=0;
errors2(i)=0;
for j=1:max_test
P=alphabet+randn(35,26)*noiselevel(i);
% 测试未经误差训练的网络
A=sim(net,P);
```

```
AA=compet(A);
errors1(i)=errors1(i)+sum(sum(abs(AA-T)))/2;
% 测试经过误差训练的网络
An=sim(netn,P);
AAn=compet(An);
errors2(i)=errors2(i)+sum(sum(abs(AAn-T)))/2;
end
end
pause
figure
plot(noise_range,errors1*100,'--',noise_range,errors2*100);
title('识别错误率');
xlabel('噪声指标');
ylabel('未经误差训练的网络 - -    经过误差训练的网络---');
%对实际含噪声的字母进行识别
for index=1:5:26
noisyJ=alphabet(:,index)+randn(35,1)*0.2;
figure;
plotchar(noisyJ);
A2=sim(net,noisyJ);
A2=compet(A2);
answer=find(compet(A2)==1);
figure;
plotchar(alphabet(:,answer));
end
```

4.9.4 结论

在这个应用中讨论了设计一个简单的模式识别系统的设计方案。需要引起注意的是，网络的训练过程不只是包含一个训练函数，它还需要用不同的输入向量进行很多次训练。

在本问题中，使用了不同的噪声向量作为输入来训练网络，使得网络具有了容错的能力，这样更适合在现实环境中使用。

4.10 径向基函数网络设计

径向基函数网络比标准的前向型 BP 网络需要更多的神经元，但是它能够按时间片来训练网络。当有很多的训练向量时，这种网络很有效果。

径向基函数网络由两层组成：第一层为隐含的径向基层，第二层为输出线性层。其网络结构如图 4-39 所示。

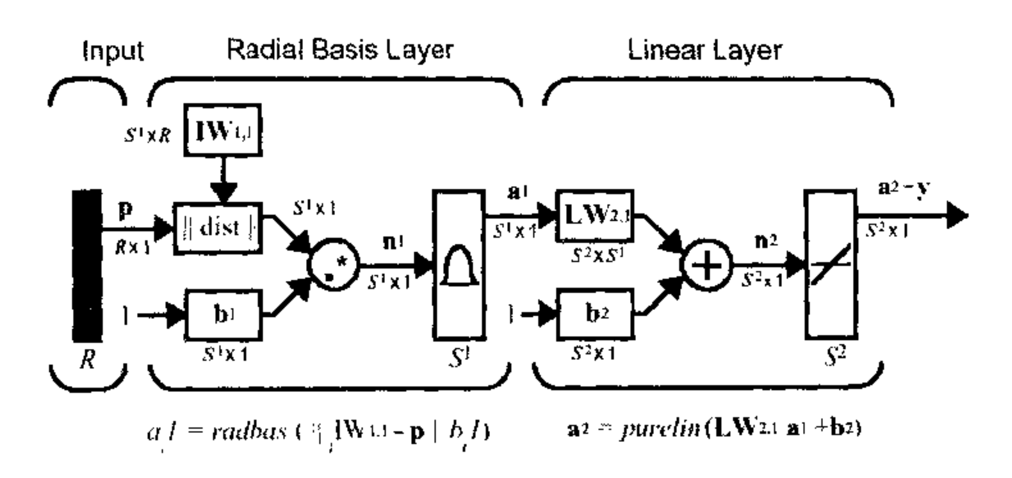


图 4-39　径向基函数网络结构

我们将设计一个径向基函数网络用于完成函数逼近的任务。

4.10.1　问题描述

已知一个函数，设计一个径向基函数神经网络来对它进行逼近。不妨设需要逼近的函数为正弦函数。

```
P = -1:.1:1;
T=sin(pi*P);
```

绘出在此正弦函数上的采样点，如图 4-40 所示。

相应的代码为：

```
plot(P,T,'+');
title('待逼近的函数样本点');
xlabel('输入值');
ylabel('目标值');
```

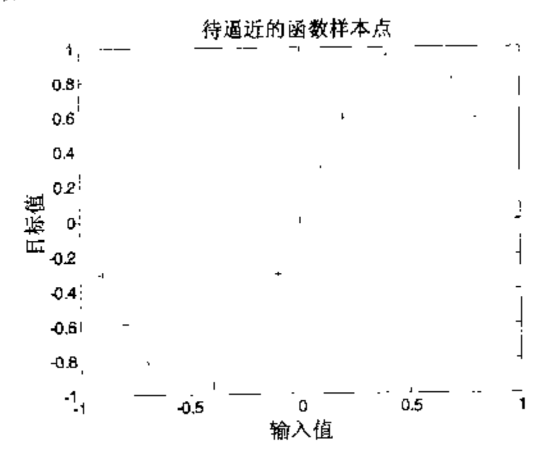


图 4-40　待逼近的函数样本点

4.10.2 网络的建立

下面是建立一个网络，使得其能够与这些已知的样本点相匹配。径向基函数网络具有两层结构：隐含层径向基神经元和输出层线性神经元。设计隐含层的径向基传递函数如下，并在图 4-41 中绘出了其曲线。

```
p = -3:.1:3;
a = radbas(p);
figure;
plot(p,a)
title('径向基传递函数');
xlabel('输入');
ylabel('输出');
```

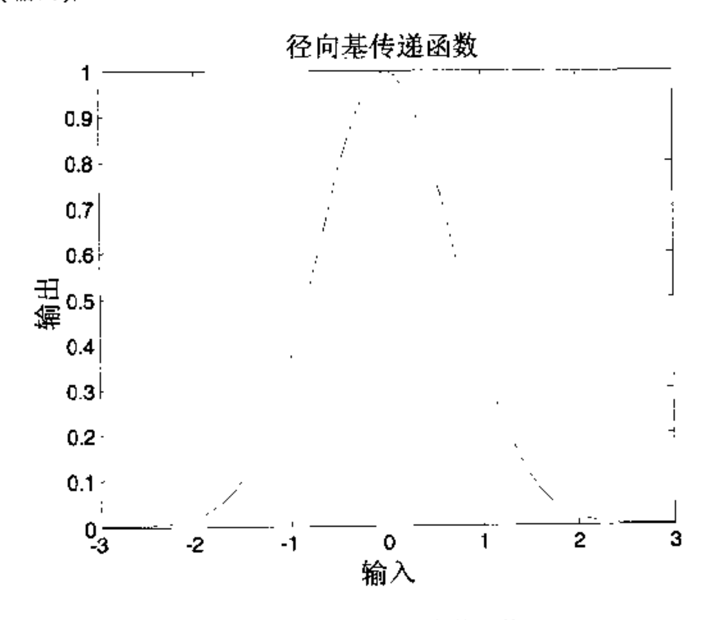


图 4-41 隐含层径向基函数

径向基函数网络隐含层中每个神经元的权重和阈值指定了相应的径向基函数的位置和宽度。每一个线性输出神经元都由这些径向基函数的加权和组成。只要每一层都有正确的权重和阈值，并且有足够的隐含层神经元，那么径向基函数网络就能够以任意的精度来逼近任意的函数。

下面给出一个示例，绘出了三个径向基函数，并且用它们的加权和产生了一个新的函数，如图 4-42 所示。虚线代表径向基函数，实线为它们的加权和函数。

```
a2 = radbas(p-1.5);
a3 = radbas(p+2);
a4 = a + a2*1 + a3*0.5;
figure;
plot(p,a,'b--',p,a2,'b--',p,a3,'b--',p,a4,'m-')
title('径向基函数的加权和');
```

```
xlabel('输入');
ylabel('输出');
```

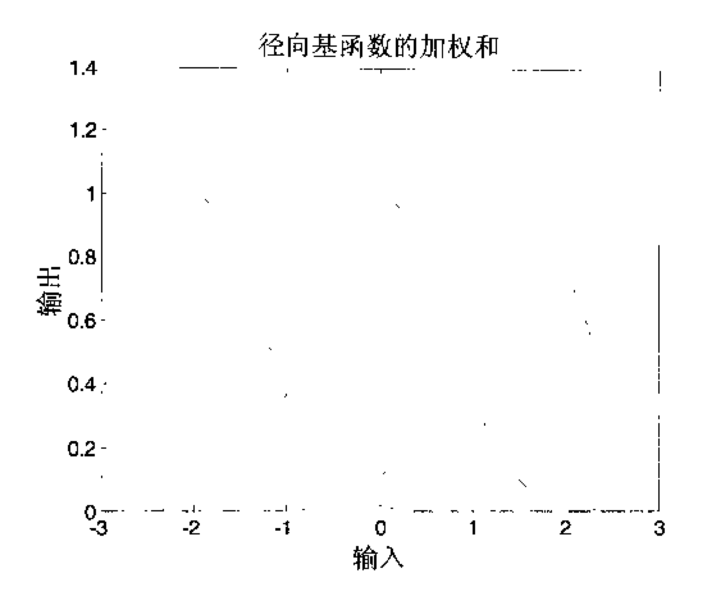


图 4-42　径向基函数及其加权和

下面使用函数 newrb 建立一个径向基函数网络来逼近由前面的样本点指定的函数。设置平方和误差参数为 0.02，展开常量为 1。

```
eg = 0.02;
sc = 1;
net=newrb(P,T,eg,sc);
```

实际上，在使用函数 newrb 建立网络时，使用了待逼近函数的样本点参数，同时也设置了目标参数，于是在建立网络时，已经完成了对网络的训练。下面对其进行仿真。

4.10.3　仿真网络

为仿真网络，首先在图中绘出待逼近函数的初始样本点。然后使用 sim 函数来产生相应的网络输出值，并将其也绘在同一幅图中，如图 4-43 所示。

```
X=-1:.01:1;
Y=sim(net,X);
hold on;
plot(X,Y);
plot(P,T,'+');
title('仿真结果');
xlabel('输入');
ylabel('网络输出及目标输出');
hold off:
```

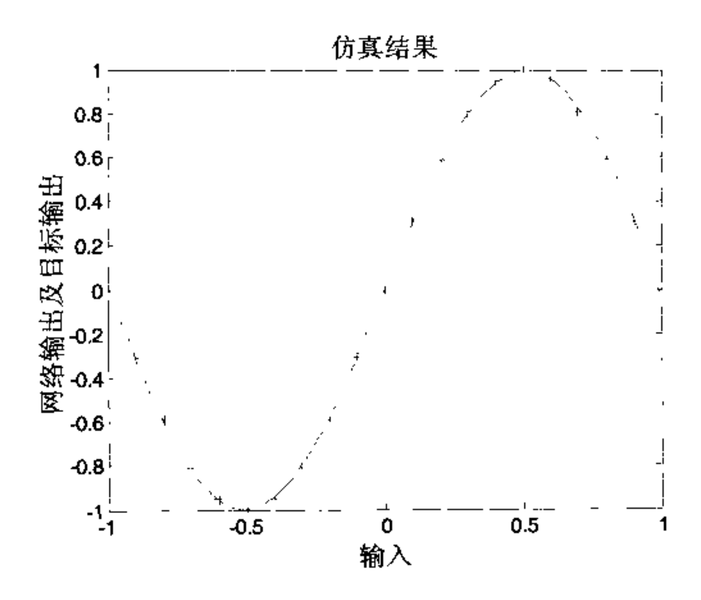


图 4-43 仿真结果及原始样本分布

从图 4-43 中可以看出，网络的仿真结果与原始待逼近的函数非常接近。这说明了使用径向基函数神经网络的效果很好。

例程 4-11 是本问题的 MATLAB 程序代码。

例程 4-11

```
%要逼近的函数样本点
P = -1:.1:1;
T=sin(pi*P);
plot(P,T,'+');
title('待逼近的函数样本点');
xlabel('输入值');
ylabel('目标值');
%径向基传递函数
p = -3:.1:3;
a = radbas(p);
figure;
plot(p,a)
title('径向基传递函数');
xlabel('输入');
ylabel('输出');
%径向基传递函数及其加权和
a2 = radbas(p-1.5);
a3 = radbas(p+2);
a4 = a + a2*1 + a3*0.5;
figure;
plot(p,a,'b--',p,a2,'b--',p,a3,'b--',p,a4,'m-')
```

```
title('径向基函数的加权和');
xlabel('输入');
ylabel('输出');
%建立网络
eg = 0.02; %设置平方和误差参数
sc = 1;     %设置展开常量
net=newrb(P,T,eg,sc);
%仿真网络并绘出结果
figure;
X=-1:.01:1;
Y=sim(net,X);
hold on;
plot(X,Y);
plot(P,T,'+');
title('仿真结果');
xlabel('输入');
ylabel('网络输出及目标输出');
hold off;
```

4.10.4 结论

使用函数 newrb 建立径向基函数神经网络，能够在给定的误差目标范围内找到能解决问题的最小的网络。

但是，即使如此，我们并不是就只需要径向基神经网络，而不需要其他的前向型网络，如 BP 网络。这是因为径向基函数网络很可能需要比 BP 网络多得多的隐含层神经元来完成工作。BP 网络使用函数 sigmoid，这样的神经元有很大的输入空间区域，而径向基函数网络使用的径向基函数输入空间区域较小。这就导致了在实际需要的输入空间较大时，需要很多的径向基神经元。

第 5 章　反馈型神经网络理论及实例

反馈网络又称递归网络，或回归网络。在反馈网络中（Feedback NNs），输入信号决定反馈系统的初始状态，然后系统经过一系列状态转移后，逐渐收敛于平衡状态。这样的平衡状态就是反馈网络经计算后的输出结果，由此可见，稳定性是反馈网络中最重要的问题之一。如果能找到网络的 Lyapunov 函数，则能保证网络从任意的初始状态都能收敛到局部最小点。Hopfield 神经网络是反馈网络中最简单且应用最广的模型，它具有联想记忆（也称 Content-addressible memory:CAM）的功能。如果可把 Lyapunov 函数定义为寻优函数的话，Hopfield 网络还可用来解决快速寻优的问题。

在这类网络中，多个神经元互连以组成一个互连神经网络，如图 5-1 所示。表示节点的状态，为节点的输入（初始值），为收敛后的输出值，有些神经元的输出被反馈至同层或前层神经元。因此，信号能够从正向和反向流通。Elman 网络和 Hopfield 神经网络是反馈网络中最有代表性的例子。

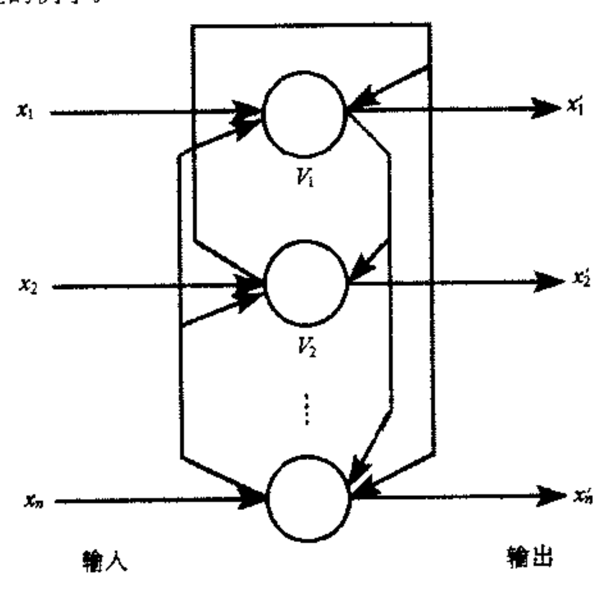


图 5-1　回归网络

本章主要内容：

- Elman 神经网络
- Hopfield 神经网络
- CG 网络模型
- 盒中脑（BSB）模型
- 双向联想记忆（BAM）

- 回归 BP 网络
- Boltzmann 机网络

5.1 Elman 神经网络

图 5-2 给出了 Elman 神经网络的结构，这种网络具有与 MLP 网络相似的多层结构。在这种网络中，除了普通的隐含层外，还有一个特别的隐含层，有时称为上下文层或状态层。该层从普通隐含层接收反馈信号，上下文层内的神经元输出被前向至隐含层。如果只有正向连接是适用的，而反馈连接被预定为恒值，那么这些网络可视为普通的前馈网络。而且，可以用 BP 算法进行训练；否则，可采用遗传算法。

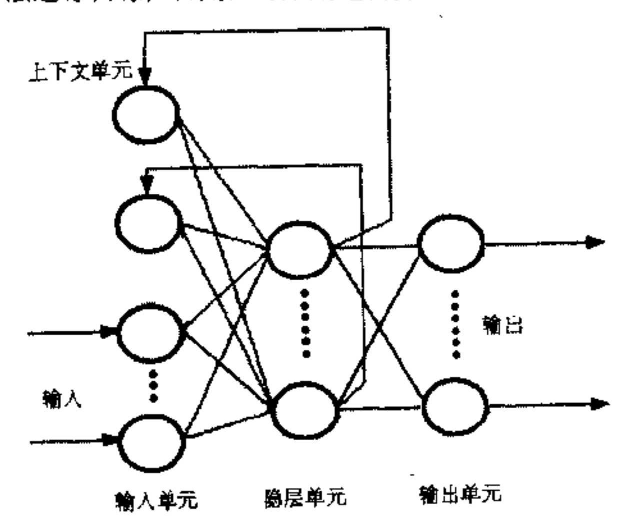


图 5-2 Elman 网络

5.2 Hopfield 神经网络

Hopfield 神经网络是美国物理学家 J.J.Hopfield 于 1982 年首先提出的。它主要用于模拟生物神经网络的记忆机理。Hopfield 神经网络有离散型和连续型两种。

5.2.1 Hopfield 神经网络的演变过程

Hopfield 神经网络的演变过程是一个非线性动力学系统，可以用一组非线性差分方程描述（对于离散型的 Hopfield 神经网络）或微分方程（Hopfield 神经网络）来描述。系统的稳定性可用所谓的“能量函数（即李雅普诺夫或哈密顿函数）”进行分析。在满足一定条件的情况下，某种“能量函数”的能量在网络运行过程中不断地减少，最后趋于稳定的平衡状态。

对于一个非线性动力学系统，系统的状态从某一初值出发经过演变后可能有如下几种结果：

- 渐进稳定点（吸引子）
- 极限环
- 混沌（chaos）
- 状态发散

因为人工神经网络的变换函数是一个有界函数，故系统的状态不会产生发散现象。目前，人工神经网络经常利用渐进稳定点来解决某些问题。例如，如果把系统的稳定点视为一个记忆的话，那么从初态朝这个稳定点的演变过程就是寻找该记忆的过程。如果把系统的稳定点视为一个能量函数的极小点，而把能量函数视为一个优化问题的目标函数，那么从初态朝这个稳定点的演变过程就是一个求解该优化问题的过程。由此可见，Hopfield 神经网络的演变过程是一种计算联想记忆或求解优化问题的过程。实际上，它的解并不需要真地去计算，而只要构成这种反馈神经网络，适当地设计其连接权和输入就可以达到这个目的。

5.2.2 离散型 Hopfield 神经网络（DHNN）

1．网络的结构及工作方式

DHNN 是一种单层的、其输入输出为二值的反馈网络，它主要用于联想记忆。DHNN 的结构如图 5-3 所示。

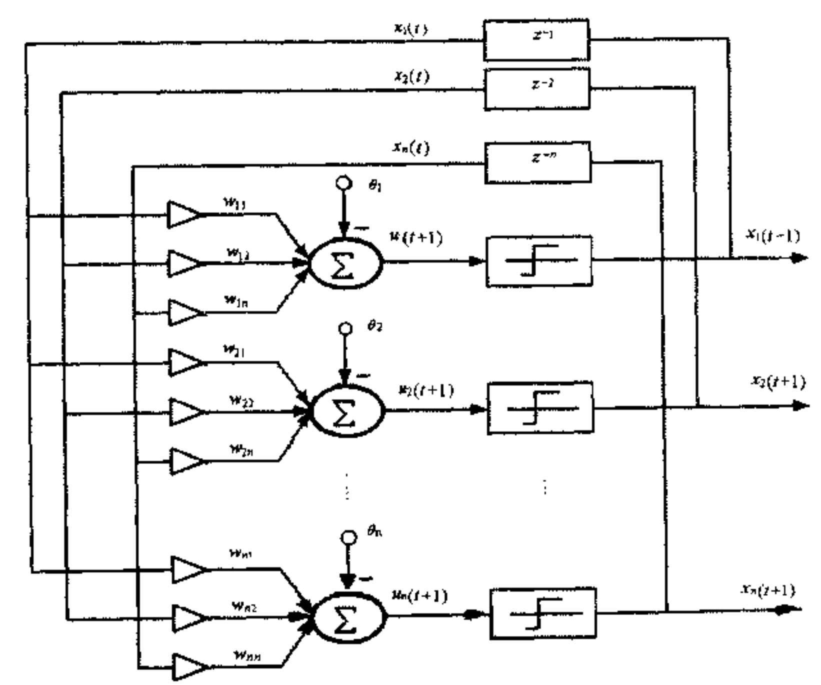


图 5-3　DHNN 结构

图 5-3 中定义 $X=[x_1,x_2,...,x_n]^T$ 为网络的状态矢量，其分量是 n 个神经元的输出，仅取+1 或－1 二值。$\theta=[\theta_1,\theta_2,...,\theta_n]^T$ 为网络的阈值矢量。$W=[w_{ij}]_{n\times n}$ 为网络的连接权矩阵，其元素 w_{ij} 表示第 j 个神经元到 i 个神经元的连接权，它为对称矩阵，即 $w_{ij}=w_{ji}$。若 w_{ii}=0，则称其网络为无自反馈的，否则，称其为有自反馈的。

DHNN 网络的计算公式如下：

$$u_i(t+1)=\sum_{j=1}^{n}w_{ij}x_j(t)-\theta_i \tag{5.1}$$

$$x_i(t+1)=\mathrm{sgn}[u_i(t+1)] \tag{5.2}$$

式中 $\mathrm{sgn}(x)$ 为符号函数：

$$\mathrm{sgn}(x)=\begin{cases}1,x\geqslant 0\\0,x<0\end{cases}$$

DHNN 主要有以下两种工作方式：

1）串行工作方式

在某一时刻只有一个神经元按照式（5.1）和式（5.2）改变状态，而其余神经元的输出保持不变。这一变化的神经元可以按照随机方式或预定的顺序来选择。例如，若达到的神经元为第 i 个，则有：

$$x_i(t+1)=\mathrm{sgn}[u_i(t+1)]$$
$$x_j(t+1)=x_j(t),(j\neq i)$$

2）并行工作方式

在某一时刻有 $N(1<N\leqslant n)$ 个神经元按照式（5.1）和式（5.2）改变状态，而其余神经元的输出保持不变。变化的这一组神经元可以按照随机方式或某种规则来选择。当 $N=n$ 时，称为全并行方式，记载某一时刻所有的神经元都按式（5.1）和式（5.2）改变状态，亦即：

$$x_i(t+1)=\mathrm{sgn}[u_i(t+1)] \quad (i=1,2,...,n)$$

若神经网络从某一状态 $X(0)$ 开始，经过有限时间 t 后，它的状态不再发生变化，这就是 DHNN 的稳定状态（吸引子）。用数学公式表示为：

$$x_i(t+1)=x_i(t)=\mathrm{sgn}(\sum_{j=1}^{n}w_{ij}x_j(t)-\theta_i) \quad (i=1,2,...,n)$$

2．DHNN 网络模型

离散的 Hopfield 神经网络模型描述如下。对于网络的每个节点：

$$x_i(k+1)=\mathrm{sgn}(\sum_{j=1}^{n}t_{ij}x_j(k)-\theta_i)\ ,\ i=1,2,...,n \tag{5.3}$$

其中
$$\operatorname{sgn}(u)=\begin{cases}+1, u \geqslant 0 \\ 0, u<0\end{cases}$$

式（5.3）的基本形式和感知器的输入/输出关系相同，这里不加讨论。

3．网络的稳定性分析

如前所述，Hopfield 神经网络的稳定性可以用网络的能量函数进行分析。DHNN 的能量函数定义为：

$$E=-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}w_{ij}x_i x_j+\sum_{i=1}^{n}\theta_i x_i$$

写成矩阵形式为：

$$E=-\frac{1}{2}X^T WX+X^T\theta$$

由于 x_i, x_j 只能为±1，θ_i 有界，能量函数是有界的，即：

$$|E|=-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}|w_{ij}||x_i||x_j|+\sum_{i=1}^{n}|\theta_i||x_i|$$

$$=\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}|w_{ij}|+\sum_{i=1}^{n}|\theta_i|$$

若从某一初始状态开始，每次迭代都能满足 $\Delta E=E(t+1)-E(t)\leqslant 0$，即网络的能量单调下降，则网络的状态最后将趋于一个稳定点（有关 DHNN 的稳定性证明，请读者参考其他一些关于神经网络的书籍）。

5.2.3 连续型 Hopfield 神经网络

连续型 Hopfield 神经网络（CHNN）是 J.J.Hopfield 于 1984 年在 DHNN 的基础上提出来的，它的原理与 DHNN 相似。由于 CHNN 是以模拟量作为网络的输入输出量，各神经元采用并行方式工作，所以它在信息处理的并行性、联想性、实时性、分布存储、协同性等方面比 DHNN 更接近于生物神经网络。

1．CHNN 网络模型

Hopfield 神经网络模型可用下列非线性微分方程描述：

$$\begin{cases} C_i \dfrac{\mathrm{d}x_i}{\mathrm{d}t} = -\dfrac{x_i}{R} + I_i + \sum\limits_{j=1}^{n} t_{ij} y_j \\ y_j = g_j(x_j) \end{cases} \tag{5.4}$$

上述模型还可用一电路来表示（如图 5-4 所示）。其中电阻 R_i 和电容 C_i 并连，模拟生物神经元输出的时间常数，跨导 t_{ij} 模拟神经元之间互连的突触特性，运算放大器用来模拟神经元的非线性特性。

定义 Hopfield 神经网络的能量函数为：

$$E = \sum_{i=1}^{n} \frac{1}{R_i} \int^{y_i} g_i^{-1}(y)\,\mathrm{d}y - \sum_{i=1}^{n} I_i y_i - \frac{1}{2} \sum_{j=1}^{n} \sum_{k=1}^{n} t_{jk} y_j y_k \tag{5.5}$$

其中 $g_i(.)$ 是 Sigmoid 函数。不难证明式（5.5）的能量函数在简单的假设下是一个 Lyapunov 函数，因此有如下的定理。

【定理 5.1】对于系统式（5.4），若 $C_i > 0$，$t_{ij} = t_{ji}$，则网络的解轨道在状态空间中总是朝着能量减小的方向运动，且网络的稳定平衡点就是 E 的极小点。

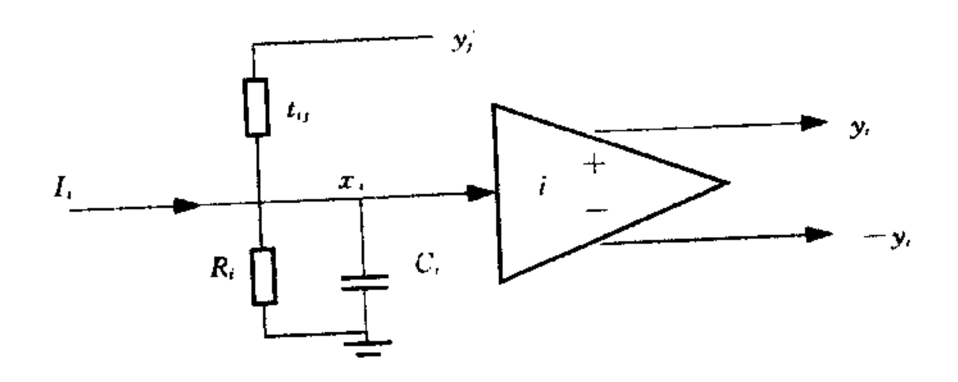


图 5-4 Hopfield 电路

当 Hopfield 神经网络用于联想记忆时，存储的信息就分布在网络的权系上。Hopfield 提出了网络在式（5.4）的 $I_i = 0$ 时，权系由记忆状态 $Y_k, k = 1,2,\ldots,m$ 构成，即：

$$t_{ij} = \sum_{k=1}^{m} Y_{ik} Y_{jk} \qquad 或 \qquad T = \sum_{k=1}^{m} Y_k Y_k^T$$

对应于式（5.4）的连续时间的 Hopfield 神经网络模型。

2．CHNN 结构图

图 5-5 所示是 CHNN 的结构图，图中每个神经元可由同向端或反向端输出（图 5-5 未画出反向端）。当由反向端输出时，它对其他神经元将起抑制作用。对于每一个神经元而

言，自己的输出信号经过其他神经元又反馈到自己，所以 CHNN 是一个连续型的非线性动力学系统。

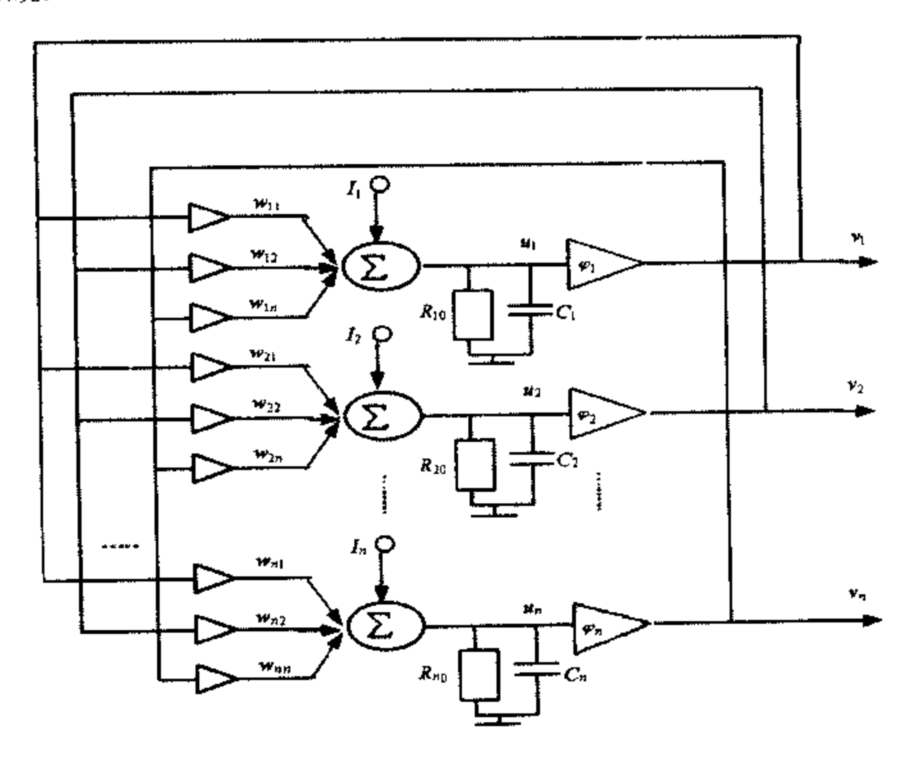


图 5-5　CHNN 结构

3．CHNN 网络稳定性分析

与 DHNN 一样，网络的稳定性分析是基于网络的能量函数。如前所述，网络的状态和输出方程式为：

$$\begin{cases} C_i \dfrac{\mathrm{d}x_i}{\mathrm{d}t} = -\dfrac{x_i}{R} + I_i + \sum\limits_{j=1}^{n} t_{ij} y_j \\ y_j = g_j(x_j) \end{cases}$$

其能量函数为：

$$E = -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij} v_i v_j - \sum_{j=1}^{n} v_i I_i + \sum_{i=1}^{n}\frac{1}{R}\int_0^{v_i}\varphi_i^{-1}(v)\,\mathrm{d}v$$

式中 $\varphi_i^{-1}(\cdot)$ 是函数 $v_i = \varphi_i(u_i)$ 的反函数。上式第三项表示输入状态和输出值关系的能量项（有关 CHNN 的稳定性证明，请读者参考其他一些有关神经网络的书籍）。

5.3　CG 网络模型

CG 网络模型可用下述一组非线性微分方程式描述：

$$\frac{\mathrm{d}x_i}{\mathrm{d}t}=a_i(x_i)\left[b_i(x_i)-\sum_{j=1}^{m}c_{ij}\,\mathrm{d}_j(x_j)\right]\qquad i=1,2,\ldots,n \tag{5.6}$$

其中由 c_{ij} 构成的连接矩阵 C 是对称矩阵，即 $c_{ij}=c_{ji}$；$a_i(.)$ 为正调函数，即 $a_i(x_i)\geqslant 0$；$d_j(.)$ 为单调函数，即 $d_j^{'}(x_i)\geqslant 0$。

在（5.6）式描述的 CG 网络模型中，x_i 代表第 i 个神经元的内部状态，$d_j(x_j)$ 是第 j 个神经元的输出，c_{ij} 是神经元 i 和 j 间耦合程度的权值，和项 $\sum_{j=1}^{n}c_{ij}d_j(x_j)$ 代表神经元 i 的输入。

关于 CG 网络模型的稳定性，我们有以下的结果。

【定理 5.2】对于系统（5.6）式，若 $a_i(x_i)\geq 0$，$d_j(x_j)$ 为单调递增，$c_{ij}=c_{ji}$，则该网络是稳定的。

5.4　盒中脑（BSB）模型

BSB（Brain-State-in-a-box）模型由下列离散方程式描述：

$$x_i(k+1)=S(x_i(k))+a\sum_{j=1}^{n}t_{ij}x_j(k)\qquad i=1,2,\ldots,n \tag{5.7}$$

其中 $t_{ij}=t_{ji}$。非线性函数 $S(.)$ 定义如下：

$$S(x)=\begin{cases}-F,x\leqslant -F\\ x,-F<\leqslant F\\ F,x>F\end{cases}$$

当系统随时间发生变化时，每个状态 x_i 逐渐趋近于 $\pm F$。事实上，但系统达到一平衡状态时，$(x_1,x_2,\ldots,x_n)$ 进入由 $(\pm F,\pm F,\ldots,\pm F)$ 构成的箱子某一角。

对应（5.7）式的连续时间模型为：

$$\frac{\mathrm{d}x_i}{\mathrm{d}t}=-x_i+S(\sum_{j=1}^{n}w_{ij}x_j) \tag{5.8}$$

其中，$w_{ij}=\sigma_{ij}+at_{ij}$，$\sigma_{ij}$ 为 Kronecker 函数，即 $\sigma_{ij}=\begin{cases}1, i=j\\0, i\neq j\end{cases}$

若定义 $y_i=\sum_{j=1}^{n}w_{ij}x_j$，将之代入（5.8）式，可得：

$$\frac{\mathrm{d}\,y_i}{\mathrm{d}t}=-y_i+\sum_{j=1}^{n}w_{ij}S(y_j)$$

由此可见，BSB 模型可看成是模型（5.6）式的一个特例，惟一不同之处是 $S(.)$ 不是处处可微的函数。

5.5　双向联想记忆（BAM）

BAM（Bidirectional Associative Memory）网络有连续时间和离散时间的两种模型。连续时间的 BAM 可由下述微分方程式描述：

$$\dot{x}_i=-a_ix_i+\sum_{j=1}^{m}t_{ij}f(y_j)+I_i \qquad i=1,2,\ldots,n$$

$$\dot{y}_j=-c_jy_j+\sum_{i=1}^{n}t_{ij}f(x_i)+J_j \qquad j=1,2,\ldots m \tag{5.9}$$

其中 a_i，c_j，I_i，J_j 为正的常数，f 为 Sigmoid 函数，$T=\lfloor t_{ij}\rfloor_{n\times m}$ 为实数矩阵。定义（5.9）式的网络能量函数为：

$$E(X,Y)=-\sum_{i=1}^{n}\sum_{j=1}^{m}t_{ij}f(x_i)f(y_j)+\sum_{i=1}^{n}a_i\int_0^{x_i}f(u_i)u_i\,\mathrm{d}u_i$$
$$+\sum_{j=1}^{m}c_j\int_0^{y_j}f(v_j)v_j\,\mathrm{d}v_j-\sum_{i=1}^{n}f(x_i)I_i-\sum_{j=1}^{m}f(y_j)J_j$$

不难证明　$\dot{E}(X,Y)=-\sum_{i=1}^{n}f(x_i)(\dot{x}_i)^2-\sum_{j=1}^{m}f(y_j)(\dot{y}_j)^2\leqslant 0$

由于 E 有界，因此对于任意初始状态 $(X(0),Y(0))$，网络将趋于一稳定状态 $(X(\infty),Y(\infty))$ 且仅当 $\dot{x}_i=\dot{y}_j=0(\forall i,\forall j)$ 时，$E(X,Y)$ 到达最小点。

离散的 BAM 类似于 Hopfield 神经网络，有下列传递函数方程式来定义：

$$x_i(k+1)=\begin{cases}1,\sum_{j=1}^{m}t_{ij}y_j(k)>\theta_i\\ x_i(k),\sum_{j=1}^{m}t_{ij}y_j(k)=\theta_i\\ 1,others\end{cases}$$

$$y_j(k+1)=\begin{cases}1,\sum_{j=1}^{m}t_{ij}x_i(k)>\eta_j\\ y_j(k),\sum_{j=1}^{m}t_{ij}x_i(k)=\eta_i\\ 1,others\end{cases}$$

对应上式的非齐次 BAM 神经网络，其能量函数可由下式定义：

$$E(X,Y)=-X^TTY+X^T\theta+Y^T\eta$$

可以证明，当状态发生变化时，E 将减小，最终达到最小点，而网络进入某一稳定状态。因此可用一个输入对 (X,Y) 回忆一个相关的双极性向量对 (X_k,Y_k)。这里的双极性是指 $X\in\{-1,1\}^n,Y\in\{-1,1\}^m$。

5.6 回归 BP 网络

由于误差方向传播（BP）算法在前向网络学习中深受欢迎，许多网络研究工作者将 BP 算法中使用的梯度下降法（Gradient decent method）应用到回归网络中，因此产生了回归 BP 算法（Recurrent back-propagation）。

回归 BP 网络可由下述非线性动态方程描述：

$$\tau\frac{\mathrm{d}z_i}{\mathrm{d}t}=-z_i+S(\sum_j w_{ij}z_j)+I_i$$

其中，z_i 代表神经元 i 的内部状态，$S(.)$ 为 Sigmoid 函数，I_i 的定义如下：

$$I_i=\begin{cases}x_i,i\in A\\ 0,others\end{cases}$$

这里 A 为输入节点的集合，x_i 为外加输入。若用 Ω 代表输出节点的集合，则隐节点是既不属于 A，也不属于 Ω 的节点，但一个节点可以同时是一输入节点（即 $I_i=x_i$），也可是一输出节点（即带有教师信号）。

网络的权矩阵$W = [W_{ij}]$通过使下列误差平方和最小来求得：

$$E = \frac{1}{2}\sum_k E_k^2$$

值得注意的是，网络的权矩阵可通过一辅助网络来修正，即：

$$\Delta W_{pq} = as'(\sum_j w_{pj} z_j) v_p z_q$$

这里v_p是下列辅助网络的稳定吸引子：

$$\frac{dv_i}{dt} = -v_i + \sum_p v_p s'(\sum_j w_{pj} z_j) w_{pj} + E_i$$

因此在权值的修正中，不必求解矩阵的逆。

5.7　Boltzmann 机网络

G.E.Hinton 和 T.J.Sejnowski 借助统计物理学的方法，对具有对称权矩阵的随机网络引进了一般的学习方法。由于这种随机网络的状态服从于统计学的 Boltzmann 分布，故被称为 Boltzmann 机。网络由可见单元和隐单元构成，每个单元只取两种状态：－1 和 1。当神经元的输入加权和发生变化时，神经元的状态随之更新，各单元之间的状态更新是异步的，可用概率来描述。神经元i的输出值取 1 的概率为：

$$P(s_i = +1) = \frac{1}{1 + \exp(-\frac{2}{T}\bar{s}_i)} \tag{5.10}$$

其中：

$$\bar{s}_i = \sum_j w_{ij} s_j$$

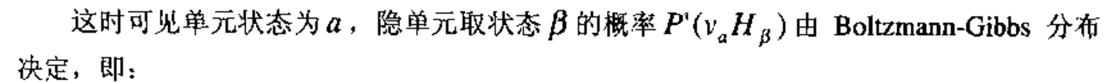
相反，神经元i的输出值取－1 的概率为：

$$P(s_i = -1) = 1 - p(s_i = +1) = \frac{1}{1 + \exp(\frac{2}{T}\bar{s}_i)}$$

这里T是网络的绝对温度。由式（5.10）可知，当输入加权和增大时，状态为 1 的概率将提高，当温度T很高时，状态取－1 和 1 的机会接近，状态容易发生变化。

网络的学习就是通过给定的一组范例，求出各单元之间连接的权值w_{ij}。Boltzmann 机学习的想法是基于统计方法。假设当网络的可见单元取状态a，隐单元取状态β时，神经元i的实际输出为$S_i^{a\beta}$，网络的能量函数可由下式定义：

$$E_{a\beta} = -\sum_i \sum_{i<j} w_{ij} S_i^{a\beta} S_j^{a\beta} = -\frac{1}{2}\sum_i \sum_j w_{ij} S_i^{a\beta} S_j^{a\beta}$$



这时可见单元状态为a，隐单元取状态β的概率$P'(v_a H_\beta)$由 Boltzmann-Gibbs 分布决定，即：

$$P'(v_a H_\beta)=\frac{1}{z}\exp(-\frac{1}{T}E_{a\beta})$$

其中：

$$z=\sum_a\sum_\beta \exp(\frac{1}{T}E_{a\beta})$$

设 Boltzmann 机由n_v个可见单元和n_h个隐单元构成，则网络共有$2^{n_v}\times 2^{n_h}=2^{(n_v+n_h)}$个状态。当温度很高时，$z$接近于网络的状态总数。实际上网络的可见单元状态处于a时的概率为：

$$P'(v_a)=\sum_\beta P'(v_a H_\beta)=\sum_\beta \frac{1}{z}\exp(-\frac{1}{T}E_{a\beta})$$

让$P(v_a)$表示网络可见单元处于a时的希望概率。Boltzmann 机的学习就是调整网络的权值，使$P'(v_a)$尽可能地逼近$P(v_a)$。

网络的温度T对 Boltzmann 机的学习很重要。如果温度太低，那么网络只能达到少数的状态，因此容易陷入局部最小点。相反，如果温度太高，陷入局部极小值的可能性虽会变小，但停止在最小值的机会也随之减少。为此，可以从高温开始，然后徐徐退火降低温度，使网络以相当高的概率收敛到最小能量的状态。这就是模拟退火法在 Boltzmann 机中应用的过程。

R.P.Lippmann 曾利用 Boltzmann 机来进行语言处理和辨识，Kohonen 等人评价了 Boltzmann 机在模式识别中的性能，他们发现 Boltzmann 机的效果要比 BP 前向网络好。

第 6 章　反馈型神经网络设计分析

前面介绍了前向型神经网络的设计，本章将介绍反馈神经网络设计。从计算的角度来看，前向型神经网络的大部分是学习网络而不具有动力学行为。反馈神经网络则是通过神经元状态的变迁而最终稳定于某一状态，从而得到联想存储或者神经计算的结果。从系统的观点看，反馈神经网络是一个非线性动力学系统。

本章主要内容包括：

- 振幅检测
- 两神经元的 Hopfield 神经网络设计
- 三神经元的 Hopfield 神经网络设计

6.1 引　　言

前面介绍了前向型神经网络的设计，本章将介绍反馈神经网络设计。从计算的角度来看，前向型神经网络的大部分是学习网络而不具有动力学行为。反馈神经网络则是通过神经元状态的变迁而最终稳定于某一状态，从而得到联想存储或者神经计算的结果。从系统的观点看，反馈神经网络是一个非线性动力学系统。

MATLAB 神经网络工具箱中主要给出了两种类型的反馈神经网络模型：Elman 神经网络和 Hopfield 神经网络。

Elman 神经网络能在有限的时间内以任意精度逼近任意函数。另外，Elman 神经网络能够存储信息以备未来使用，所以它不仅能够学习空域模式，也能够学习时域模式。在本章的第 2 节，我们就要利用它的这一特点，设计一个 Elman 神经网络，用于对输入波形进行振幅检测。

Hopfield 神经网络是一种非常经典的反馈型神经网络。常用于存储一个或者多个稳定的目标向量，当向网络提供近似的向量时，这些向量就像是被存储在网络中的一样，会被唤起。Hopfield 神经网络设计的目标就是使得网络存储一些特定的平衡点，给定网络一个初始条件，网络经过调整后会在这样的点上停下来。由于输出又被反馈到输入，所以一旦网络开始运行，整个网络就是递归的。在本章的第 3 节和第 4 节将分别设计一个两神经元和一个三神经元的 Hopfield 神经网络。

6.2 振 幅 检 测

Elman 神经网络通常由两层网络结构，它存在从第一层的输出到第一层输入的反馈，这种反馈连接的结构使得其被训练后不仅能够识别和产生空域模式，还能够识别和产生时

域模式。其网络的两层结构如图 6-1 所示。

Elman 神经网络的这种组合结构特点使得其能在有限的时间内以任意精度逼近任意函数。这一点只需要通过给递归层设置足够多的神经元来实现。当需要逼近的函数复杂性增加时，需要的递归层神经元数目也要增加。

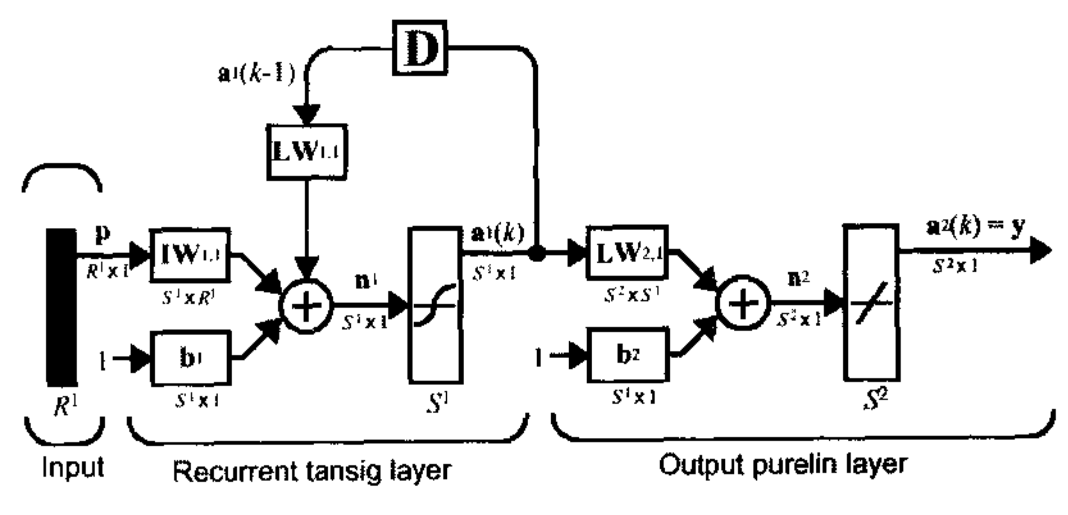


图 6-1　Elman 神经网络结构

正是由于 Elman 神经网络能够存储信息以备未来使用，所以它不仅能够学习空域模式，也能学习时域模式。在这里，我们就要利用其这一特点，进行振幅检测。所谓振幅检测，就是对时域模式进行识别，然后按空域模式对其进行分类。

振幅检测需要在一段时间里对网络提供一个波形，这样网络就会输出这个波形的振幅。虽然这并不是一个很难的问题，但是能够反映出 Elman 神经网络的设计过程。下面我们就来设计这样一个网络。

6.2.1　问题描述

首先定义两个正弦波形函数，一个振幅为 1，另一个振幅为 2。

```
p1=sin(1:20);
p2=sin(1:20)*2;
```

上述波形的目标输出为它们的振幅。

```
t1=ones(1,20);
t2=ones(1,20)*2;
```

将上述波形组合成一个序列，使得每个波形出现两次。我们使用这个组合后的延长了的波形信号来训练 Elman 神经网络。

```
p=[p1 p2 p1 p2];
t=[t1 t2 t1 t2];
```

我们希望输入和输出信号都是序列，故使用转换函数 con2seq 来把它们从矩阵信号转

换成序列信号。图 6-2 中绘出了输入信号和目标信号的曲线。

```
Pseq=con2seq(p);
Tseq=con2seq(t);
```

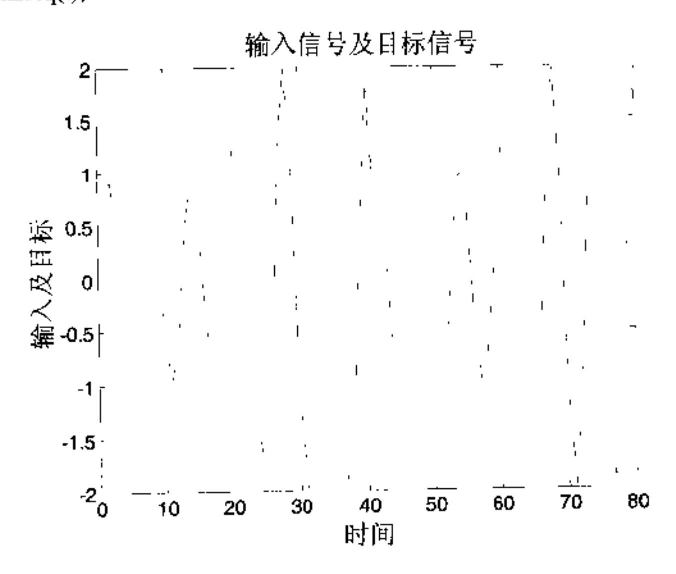


图 6-2 输入信号及目标信号曲线

6.2.2 网络初始化

本问题需要使用 Elman 神经网络检测一个单输入值（即输入波形信号），并且在每一个时间步长上输出一个单值信号（即振幅值）。因此，网络必须有一个输入元件和一个输出神经元。

```
R=1;  % 一个输入元件
S2=1;  % 第二层一个输出神经元
```

从原理上说，递归网络层可以包含任意数目的神经元。不过，随着问题复杂程度的增加，需要更多的神经元数目才能很好地工作。

我们在这里要讨论的问题比较简单，所以在第一层中只要设计 10 个递归神经元就足够解决问题了。

```
S1=10;% 在第一层中包含 10 个递归神经元单元
```

现在可以使用 newelm 函数来建立网络，初始化权重矩阵和阈值向量。产生的网络可以有一个其值在－2～2 之间变化的输入。在本问题中，我们使用 traingdx 函数作为学习速率，它能根据梯度下降的情况，以及自适应学习的速率来调整权重和阈值。

```
net=newelm([-2 2],[S1 S2],{'tansig','purelin'},'traingdx');
```

6.2.3 网络训练

现在开始训练网络，首先设置训练时间为 500 个时间单位（epoch），然后就可以使用 train 函数来进行网络的训练。

```
net.trainParam.epochs=500;
[net,tr]=train(net,Pseq,Tseq);
```

在利用 train 函数对网络进行训练执行以上代码后，在 MATLAB 命令行中将实时地显示出网络的训练状态。如下所示：

```
TRAINGDX,Epoch 0/500, MSE 9.98807/0, Gradient 19.1745/1e-006
TRAINGDX,Epoch 25/500, MSE 0.250986/0, Gradient 0.290913/1e-006
TRAINGDX,Epoch 50/500, MSE 0.246709/0, Gradient 0.0725029/1e-006
TRAINGDX,Epoch 75/500, MSE 0.241614/0, Gradient 0.0666679/1e-006
TRAINGDX,Epoch 100/500, MSE 0.221712/0, Gradient 0.0627453/1e-006
TRAINGDX,Epoch 125/500, MSE 0.0865298/0, Gradient 0.127753/1e-006
TRAINGDX,Epoch 150/500, MSE 0.0721242/0, Gradient 0.252093/1e-006
TRAINGDX,Epoch 175/500. MSE 0.068302/0, Gradient 0.067186/1e-006
TRAINGDX,Epoch 200/500, MSE 0.0652771/0, Gradient 0.0415012/1e-006
TRAINGDX,Epoch 225/500, MSE 0.0529941/0, Gradient 0.0397959/1e-006
TRAINGDX,Epoch 250/500, MSE 0.034749/0, Gradient 0.272335/1e-006
TRAINGDX,Epoch 275/500, MSE 0.0326495/0, Gradient 0.123384/1e-006
TRAINGDX,Epoch 300/500, MSE 0.0306567/0, Gradient 0.0397829/1e-006
TRAINGDX,Epoch 325/500, MSE 0.0265246/0, Gradient 0.0250898/1e-006
TRAINGDX,Epoch 350/500, MSE 0.0194044/0, Gradient 0.0873426/1e-006
TRAINGDX,Epoch 375/500, MSE 0.0190188/0, Gradient 0.0479671/1e-006
TRAINGDX,Epoch 400/500, MSE 0.0187596/0, Gradient 0.0201117/1e-006
TRAINGDX,Epoch 425/500, MSE 0.0180997/0, Gradient 0.0176995/1e-006
TRAINGDX,Epoch 450/500, MSE 0.0165411/0, Gradient 0.0154307/1e-006
TRAINGDX,Epoch 475/500,MSE 0.0166451/0, Gradient 0.27754/1e-006
TRAINGDX,Epoch 500/500, MSE 0.0155302/0, Gradient 0.0724595/1e-006
```

图 6-3 显示了训练过程中的误差变化情况。

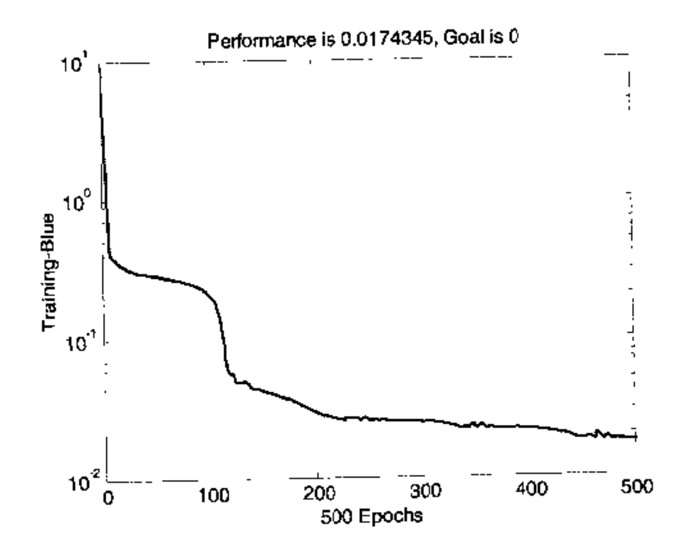


图 6-3 训练过程中的误差变化

从误差曲线及 MATLAB 命令行实现的结果来看，在第 500 个时间单位的时候，平均

平方误差为 0.015 530 2。下面测试网络。

6.2.4　网络测试

我们用原始输入 Pseq 作为输入来测试网络。使用函数 simuelm 来仿真网络。

```
a=sim(net,Pseq);
```

上面得到结果为网络输出的 Pseq 信号的振幅，在同一幅图上绘出网络测试得到的结果 a，以及目标信号 Tseq 的曲线，如图 6-4 所示。

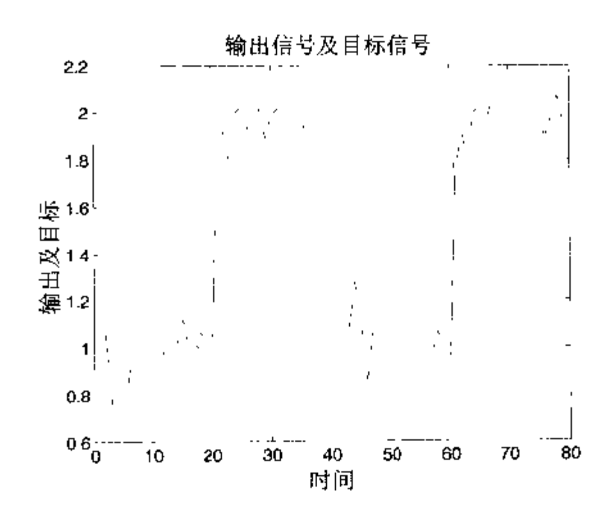


图 6-4　测试输出信号及目标信号曲线

从图 6-4 的曲线可以看出，输出信号在目标信号的两侧有小幅度的振荡，但能跟着输入信号的振幅变化而变化，网络能较好地检测出输入信号的振幅。这说明我们设计的网络是比较成功的。当我们增加递归层的神经元单元的数目或者是延长训练时间时，都能得到更好的性能。

例程 6-1 是本问题的 MATLAB 程序代码。

例程 6-1

```
%定义输入信号及目标信号
Time=1:80;
p1=sin(1:20);
p2=sin(1:20)*2;
t1=ones(1,20);
t2=ones(1,20)*2;
p=[p1 p2 p1 p2];
t=[t1 t2 t1 t2];
Pseq=con2seq(p);
Tseq=con2seq(t);
%绘出输入信号及目标信号曲线
figure;
```

```
plot(Time,cat(2,Pseq{:}),'--',Time,cat(2,Tseq{:}));
xlabel('时间');
ylabel('输入及目标');
title('输入信号及目标信号');
%开始生成网络
R=1; % 一个输入元件
S2=1; % 第二层一个输出神经元
S1=10;% 在第一层中包含 10 个递归神经元单元
net=newelm([-2 2],[S1 S2],{'tansig','purelin'},'traingdx');
%训练网络
net.trainParam.epochs = 500; %设置训练时间为 500 个时间单位
[net,tr]=train(net,Pseq,Tseq);
%开始测试网络性能
a = sim(net,Pseq);
%绘出输出信号及目标信号曲线
figure;
plot(Time,cat(2,a{:}),Time,cat(2,Tseq{:}),'--');
xlabel('时间');
ylabel('输出及目标');
title('输出信号及目标信号');
```

6.2.5 网络的推广应用及完善

对于前面我们设计的网络，虽然能够检测到被其训练过的波形信号的振幅，但是对于那些振幅没有被训练过的正弦信号，它不一定能够准确地检测出来。

下面我们就来进行这项测试工作。首先定义一个新的信号，它由两个振幅分别为 1.6 和 1.2 的正弦信号分别重复两次构成。

```
p3=sin(1:20)*1.6;
t3=ones(1,20)*1.6;
p4=sin(1:20)*1.2;
t4=ones(1,20)*1.2;
pg=[p3 p4 p3 p4];
tg=[t3 t4 t3 t4];
pgseq=con2seq(pg);
```

在这里用输入信号序列 pg 和目标信号序列 tg 来测试我们刚才设计的网络推广到新的振幅信号的能力。

再次调用函数 sim 来仿真 Elman 神经网络，结果如图 6-5 所示。

```
a2=sim(net,pgseq);
```

这一次网络得到的输出结果不太令人满意。从曲线上可以看出，网络的输出结果好像能跟踪输入信号的振幅，在其两侧振荡，但是振荡的幅度太大，使得其结果已经变得很不精确。

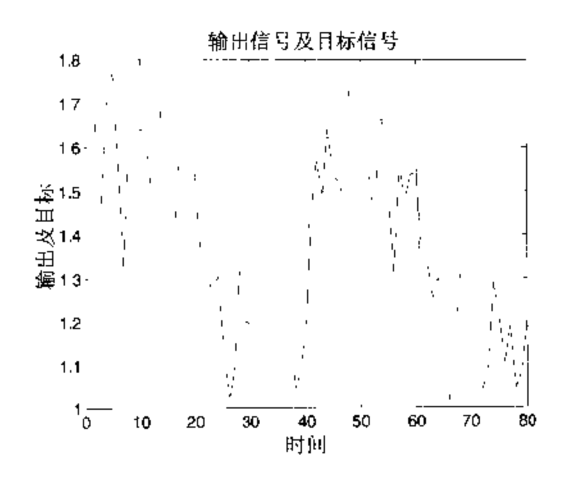


图 6-5　新的振幅信号检测结果及目标

为了改善这种状况，增加网络的推广能力，可以利用更多的振幅信号来训练网络，而不是像本网络的设计只用了 1.0 和 2.0 两种振幅信号。使用 3～4 种不同振幅、不同波形的信号来进行训练，将会有更好的效果。有兴趣的读者可以自己试着改变本网络的许多参数，继续完善这个网络，使得它的检测能力更强。

6.3　两神经元的 Hopfield 神经网络设计

Hopfield 神经网络常用于存储一个或者多个稳定的目标向量，当向网络提供近似的向量时，这些向量就像是被存储在网络中的一样，会被唤起。

Hopfield 神经网络设计的目标就是使得网络存储一些特定的平衡点，当给定网络一个初始条件时，网络最后会在这样的点上停下来。由于输出又被反馈到输入，所以一旦网络开始运行，整个网络就是递归的。Hopfield 神经网络很有可能在原始设计好的点上运行。

Hopfield 神经网络的结构如图 6-6 所示。

6.3.1　问题描述

考虑一个具有两个神经元的 Hopfield 神经网络。每个神经元有一个阈值和一个权重，这样刚好与具有两个元素的输入向量相匹配。定义存储在网络中的目标平衡点为矩阵 ***T*** 的两个列向量。

```
T = [+1 -1;-1 +1]'
T =
1     -1
-1     1
```

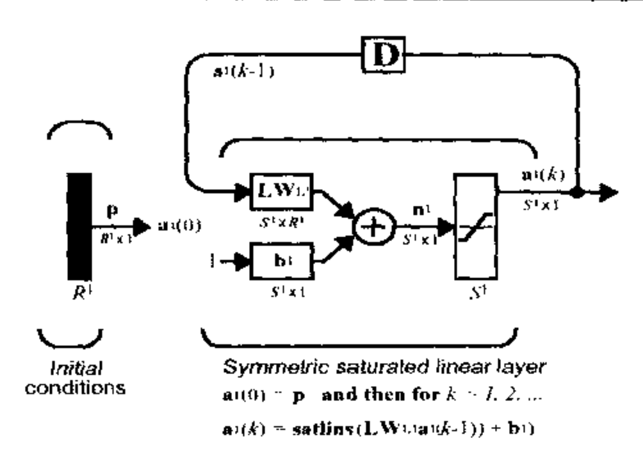


图 6-6 Hopfield 神经网络结构

图 6-7 绘出了 Hopfield 状态空间图，其中用“*”标记了上述的两个稳定点。

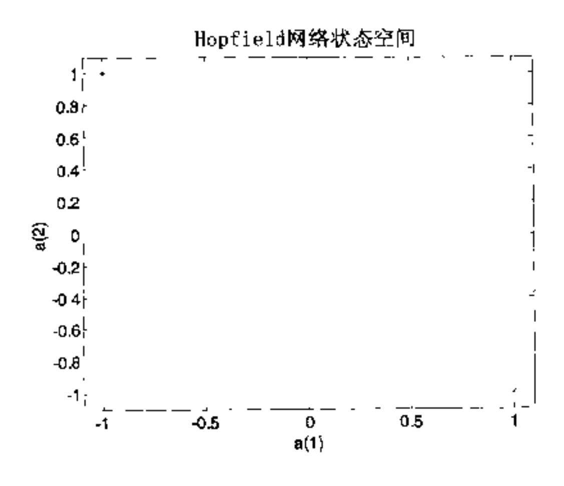


图 6-7 稳定点

```
plot(T(1,:),T(2,:),'r*');
axis([-1.1 1.1 -1.1 1.1]);
title('Hopfield 神经网络状态空间');
xlabel('a(1)');
ylabel('a(2)');
```

6.3.2 建立网络

使用 newhop 函数来建立网络。

```
net=newhop(T);
```

该函数将返回每个神经元的权值和阈值。可用下面的代码得到结果。

```
W=net.LW{1,1}
b=net.b{1,1}
```

得到的结果为：

```
W =
0.6925    -0.4694
-0.4694    0.6925
b =
0
0
```

6.3.3　网络的测试

下面进行网络的测试。

1．测试一

首先，使用目标向量 ***T*** 来测试其是否被存储在网络中。使用 sim 函数，并将目标向量作为输入。

```
[Y,Pf,Af] = sim(net,2,[],T);
Y
```

得到的结果为：

```
Y =
1    -1
-1    1
```

与我们所设想的一样，网络的输出就是目标向量。

2．测试二

下面继续测试目标向量的稳定性。我们给网络输入一个随机向量，并且对网络做 20 步仿真，测试其输出结果。我们希望输出值为网络的某个稳定点。在网络状态空间图中显示了其到达目标点的轨迹，如图 6-8 所示。

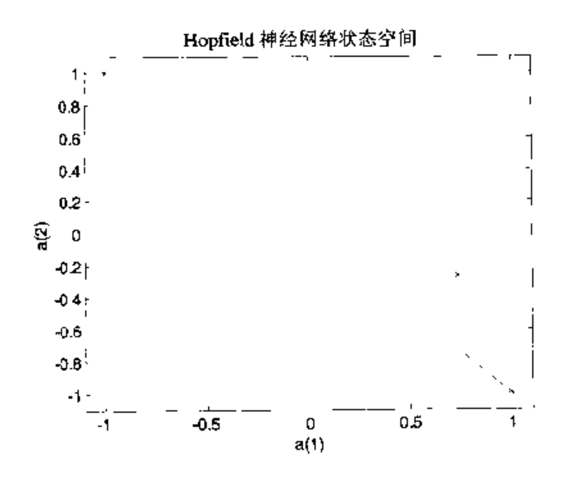


图 6-8　从一个随机点求解的轨迹

```
a={rands(2,1)};
[y,Pf,Af]=sim(net,{1 20},{},a);
figure;
```

```
plot(T(1,:),T(2,:),'r*');
axis([-1.1 1.1 -1.1 1.1]);
title('Hopfield 神经网络状态空间');
xlabel('a(1)');
ylabel('a(2)');
record=[cell2mat(a) cell2mat(y)];
start=cell2mat(a);
hold on
plot(start(1,1),start(2,1),'bx',record(1,:),record(2,:));
```

从图 6-8 可以看出，网络输出最终到达了左上角的稳定点，这与我们希望得到的结果相吻合。

3．测试三

为了更有力地说明 Hopfield 神经网络对目标点的跟踪能力，我们选择了 25 个随机起始点来完成上面刚做的测试工作。

```
color = 'rgbmy';
axis([-1.1 1.1 -1.1 1.1]);
title('Hopfield 神经网络状态空间');
xlabel('a(1)');
ylabel('a(2)');
hold on
for i=1:25
a = {rands(2,1)};
[y,Pf,Af] = sim(net,{1 20},{},a);
record=[cell2mat(a) cell2mat(y)];
start=cell2mat(a);
plot(start(1,1),start(2,1),'kx',record(1,:),
record(2,:),color(rem(i,5)+1))
drawnow
end;
```

在图 6-9 中绘出了这些点到达目标点的轨迹。可以看出，这些点都能够回到稳定点。

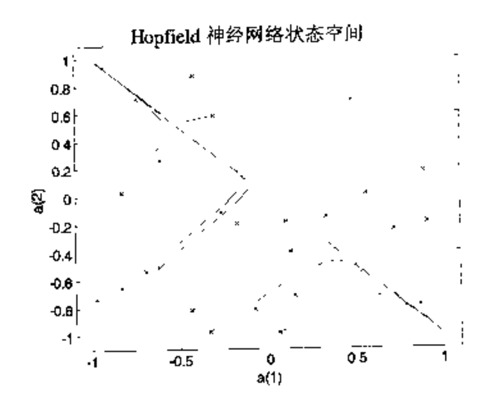


图 6-9　多个随机起始点到达目标点的轨迹



当初始点离左上角的稳定点较近时，其最终就会达到左上角的稳定点。相反，当初始点离右下角的稳定点较近时，也就会到达右下角的稳定点。正是由于具备达种对一个初始输入能够找到其相应的最近的存储值的能力，使得 Hopfield 神经网络模型十分有用。

4．测试四

下面考虑给网络输入特殊初始权重，进行仿真的情况。

```
axis([-1.1 1.1 -1.1 1.1]);
title('Hopfield 神经网络状态空间');
xlabel('a(1)');
ylabel('a(2)');
hold on
plot(0,0,'ko');
P = [-1.0 -0.5 0.0 +0.5 +1.0;   -1.0 -0.5 0.0 +0.5 +1.0];
color = 'rgbmy';
for i=1:5
a = {P(:,i)};
[y,Pf,Af] = sim(net,{1 50},{},a);
record=[cell2mat(a) cell2mat(y)];
start=cell2mat(a);
plot(start(1,1),start(2,1),'kx',record(1,:),
record(2,:),color(rem(i,5)+1))
drawnow
end
```

相应的轨迹如图 6-10 所示。

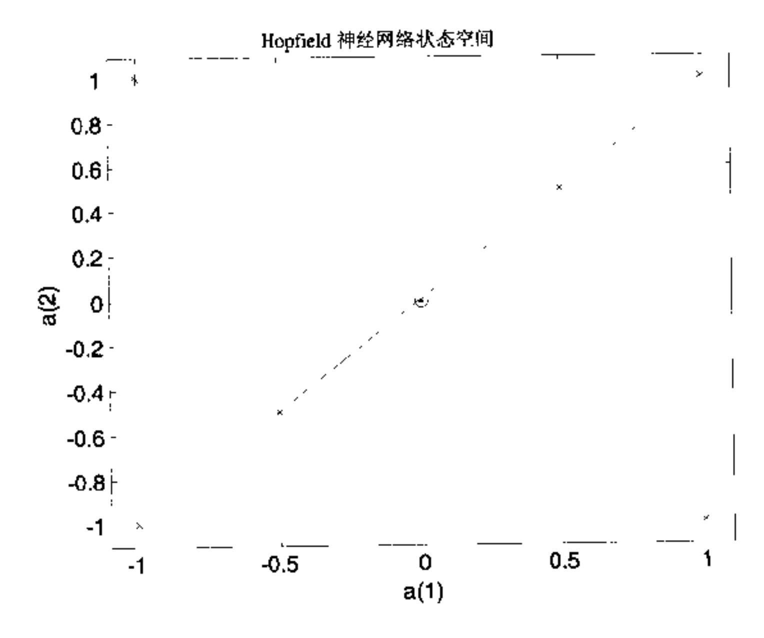


图 6-10　指定特殊初始点仿真得到的轨迹

从图 6-10 可以观察到一个奇怪的现象：从我们刚才指定的特殊初始点出发的权重，在经过网络调整以后，并不回到我们期望的目标点。它们都回到一个特定点，这个点位于我们的两个目标点的连线的正中间。

这样看起来好像是整个网络中又增加了一个稳定点。实际上，这并不是一个真正的稳定点。



我们指定的这些特定初始权重都位于目标点连线的中垂线上，才会有这样的结果。当这些初始点由于噪声影响，对中垂线稍有偏移时，再经过仿真其结果就不会是这个新增的稳定点。所以说这个点并不是真正的稳定点。关于这个现象，在设计网络时应该加以注意。

例程 6-2 是本问题的 MATLAB 程序代码。

例程 6-2

```
%定义存储在网络中的目标平衡点
T = [+1 -1;-1 +1]';
plot(T(1,:),T(2,:),'r*');
axis([-1.1 1.1 -1.1 1.1]);
title('Hopfield 神经网络状态空间');
xlabel('a(1)');
ylabel('a(2)');
%建立网络，并得到权重和阈值
net=newhop(T);
W=net.LW{1,1}
b=net.b{1,1}
%使用原始平衡点仿真网络
[Y,Pf,Af] = sim(net,2,[],T);
Y
%使用一个随机点仿真网络，并绘出其到达稳定点的轨迹
a = {rands(2,1)};
[y,Pf,Af] = sim(net,{1 20},{},a);
figure;
plot(T(1,:),T(2,:),'r*');
axis([-1.1 1.1 -1.1 1.1]);
title('Hopfield 神经网络状态空间');
xlabel('a(1)');
ylabel('a(2)');
record=[cell2mat(a) cell2mat(y)];
start=cell2mat(a);
hold on
plot(start(1,1),start(2,1),'bx',record(1,:),record(2,:));
%用多个随机点仿真网络，并绘出相应的轨迹
figure;
color = 'rgbmy';
```



```
axis([-1.1 1.1 -1.1 1.1]);
title('Hopfield 神经网络状态空间');
xlabel('a(1)');
ylabel('a(2)');
hold on
for i=1:25
a = {rands(2,1)};
[y,Pf,Af] = sim(net,{1 20},{},a);
record=[cell2mat(a) cell2mat(y)];
start=cell2mat(a);
plot(start(1,1),start(2,1),'kx',record(1,:),record(2,:),
color(rem(i,5)+1));
drawnow
end;
%给网络指定特殊初始权重进行仿真，得到相应的轨迹
figure;
plot(T(1,:),T(2,:),'r*');
axis([-1.1 1.1 -1.1 1.1]);
title('Hopfield 神经网络状态空间');
xlabel('a(1)');
ylabel('a(2)');
hold on
plot(0,0,'ko');
P = [-1.0 -0.5 0.0 +0.5 +1.0;   -1.0 -0.5 0.0 +0.5 +1.0];
color = 'rgbmy';
for i=1:5
a = {P(:,i)};
[y,Pf,Af] = sim(net,{1 50},{},a);
record=[cell2mat(a) cell2mat(y)];
start=cell2mat(a);
plot(start(1,1),start(2,1),'kx',record(1,:),
record(2,:),color(rem(i,5)+1))
drawnow
end
```

6.4　三神经元的 Hopfield 神经网络设计

6.3 节介绍了二神经元的 Hopfield 神经网络设计，本节再介绍三神经元的 Hopfield 神经网络设计实例。整个设计过程与 6.3 节基本相同。

6.4.1　问题描述

考虑一个具有三个神经元的 Hopfield 神经网络。每个神经元有一个阈值和一个权重，这样刚好与具有三个元素的输入向量相匹配。定义存储在网络中的目标平衡点为矩阵 T 的

两个三维列向量。

```
T = [+1 +1;-1 +1;-1 -1]
T =
1      1
-1     1
-1    -1
```

图 6-11 绘出了 Hopfield 神经网络状态空间图，其中用“*”标记了上述的两个稳定点。由于为三维空间，为了便于观察，我们在绘图时进行了必要的设置。所有的状态都将被限制在图中的框中。

```
axis([-1 1 -1 1 -1 1]');
set(gca,'box','on');
axis manual;
hold on;
plot3(T(1,:),T(2,:),T(3,:),'r*');
title('Hopfield 神经网络状态空间');
xlabel('a(1)');
ylabel('a(2)');
zlabel('a(3)');
view([37.5 30]);
```

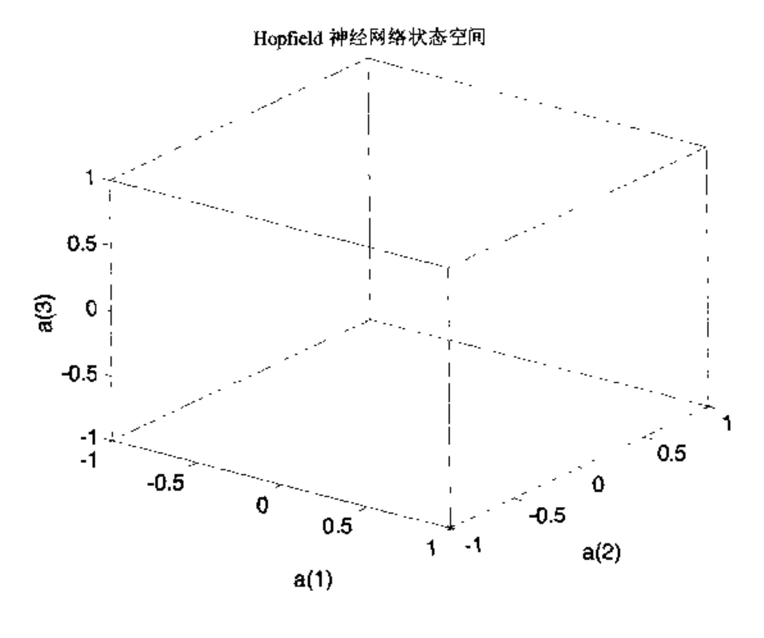


图 6-11 稳定点

6.4.2 建立网络

同样，使用 newhop 函数来建立网络。该函数使用给定的稳定点来建立网络。

```
net=newhop(T);
```

该函数将返回每个神经元的权值和阈值。可用下面的代码得到结果。

```
W=net.LW{1,1}
b=net.b{1,1}
```

得到的结果为：

```
W =
0.2231        0        0
0       1.1618        0
0        0      0.2231
b =
0.8546
0
-0.8546
```

6.4.3 网络的测试

下面测试目标向量的稳定性。

1．测试一

我们给网络输入一个随机向量，并且对网络做 50 步仿真，测试其输出结果。我们希望输出值为网络的某个稳定点。在图 6-12 所示的神经网络状态空间图中显示了其到达目标点的轨迹。

```
a = {rands(3,1)};
[y,Pf,Af] = sim(net,{1 10},{},a);
figure;
axis([-1 1 -1 1 -1 1])';
set(gca,'box','on');
axis manual;
hold on;
plot3(T(1,:),T(2,:),T(3,:),'r*');
record=[cell2mat(a) cell2mat(y)];
start=cell2mat(a);
plot3(start(1,1),start(2,1).start(3,1),'bx',record(1,:),
record(2,:),record(3,:));
view([37.5 30]);
title('Hopfield 神经网络状态空间');
xlabel('a(1)');
ylabel('a(2)');
zlabel('a(3)');
```

从图 6-12 以看出，网络输出最终到达了稳定点，这与我们希望得到的结果相吻合。

2．测试二

为了更有力地说明 Hopfield 神经网络对目标点的跟踪能力，我们选择了 25 个随机起始点，来完成上面刚做的测试工作。

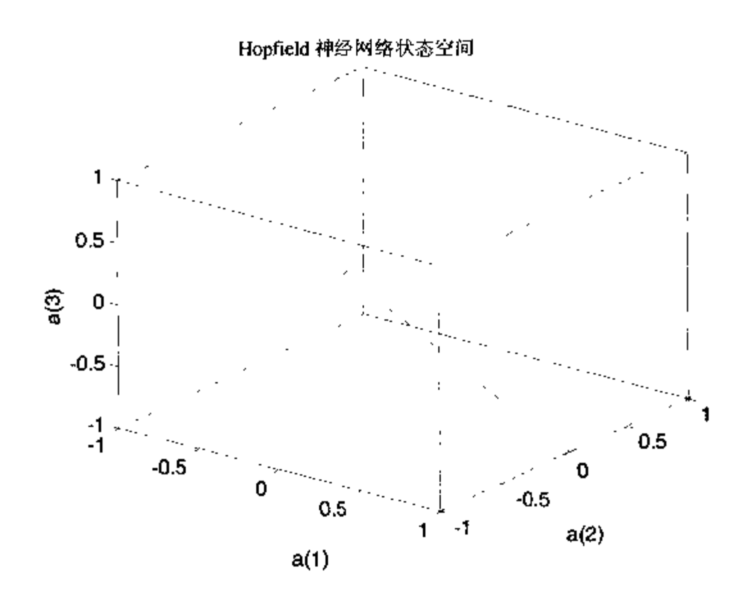


图 6-12　从一个随机点求解的轨迹

为完成测试二，我们有如下代码：

```
axis([-1 1 -1 1 -1 1]');
set(gca,'box','on');
axis manual;
hold on;
view([37.5 30]);
plot3(T(1,:),T(2,:),T(3,:),'r*');
color = 'rgbmy';
for i=1:25
a = {rands(3,1)};
[y,Pf,Af] = sim(net,{1 10},{},a);
record=[cell2mat(a) cell2mat(y)];
start=cell2mat(a);
plot3(start(1,1),start(2,1),start(3,1),'kx',
record(1,:),record(2,:),record(3,:),
color(rem(i,5)+1));
drawnow
end
title('Hopfield 神经网络状态空间');
xlabel('a(1)');
ylabel('a(2)');
zlabel('a(3)');
```

在图 6-13 中绘出了这些点到达目标点的轨迹。

可以看出，这些点都能够回到稳定点。需要注意的是，当初始点离左上角的稳定点较近时，其最终就会达到左上角的稳定点。相反，当初始点离右上角的稳定点较近时，也就会到达右上角的稳定点。

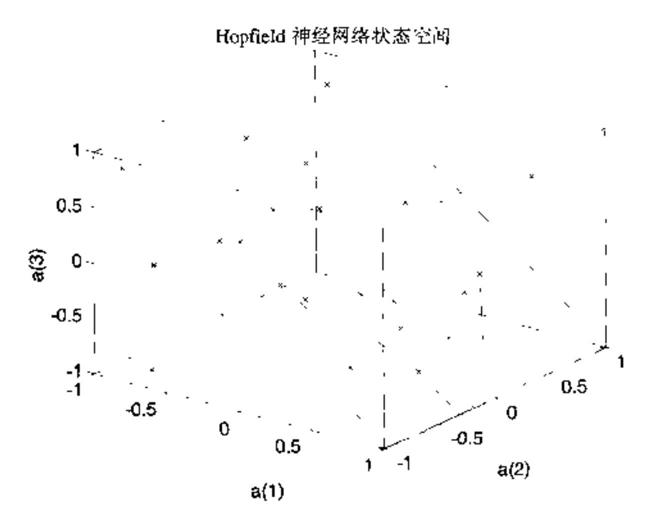


图 6-13 多个随机起始点到达目标点的轨迹

3. 测试三

下面我们指定一些点，使用设计的 Hopfield 神经网络来进行仿真。

```
axis([-1 1 -1 1 -1 1])';
set(gca,'box','on');
axis manual;
hold on;
P = [ 1.0   -1.0   -0.5   1.00   1.00   0.0; 0.0    0.0    0.0
0.00   0.00 -0.0;-1.0    1.0    0.5 -1.01 -1.00   0.0];
view([37.5 30]);
color ='rgbmy';
for i=1:6
a = {P(:,i)};
[y,Pf,Af] = sim(net,{1 10},{},a);
record=[cell2mat(a) cell2mat(y)];
start=cell2mat(a);
plot3(start(1,1),start(2,1),start(3,1),'kx',
record(1,:),record(2,:),record(3,:),
color(rem(i,5)+1))
drawnow
end
title('Hopfield 神经网络状态空间');
xlabel('a(1)');
ylabel('a(2)');
zlabel('a(3)');
```



指定的点正好在两个目标稳定点的中间。仿真得到的结果如图 6-14 所示。从图中可以看出，从这些初始点出发的点，最后都到达了两个目标稳定点的中间点上。这不是我们希望得到的稳定点。

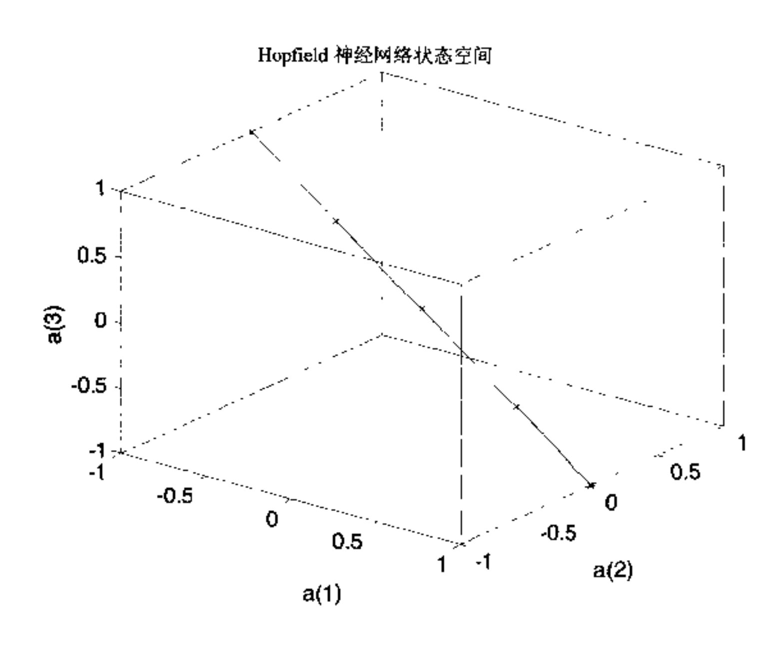


图 6-14 指定特殊初始点的仿真轨迹

在上一节我们就提到过这种现象，这并不是一个真正稳定点。当这些初始点稍有误差时，其经过调整后就不会再回到这个点上。

例程 6-3 是本问题的 MATLAB 程序代码。

例程 6-3

```
%指定存储在网络中的目标平衡点
T = [+1 +1;-1 +1; -1 -1]
axis([-1 1 -1 1 -1 1])';
set(gca,'box','on');
axis manual;
hold on;
plot3(T(1,:),T(2,:),T(3,:),'r*');
title('Hopfield 神经网络状态空间');
xlabel('a(1)');
ylabel('a(2)');
zlabel('a(3)');
view([37.5 30]);

%建立网络，并得到权重和阈值
net=newhop(T);
W=net.LW{1.1}
b=net.b{1,1}

%使用一个随机点仿真网络，并绘出其到达稳定点的轨迹
a = {rands(3,1)};
```

```
[y,Pf,Af] = sim(net,{1 10},{},a);
figure;
axis([-1 1 -1 1 -1 1])';
set(gca,'box','on');
axis manual;
hold on;
plot3(T(1,:),T(2,:),T(3,:),'r*');
record=[cell2mat(a) cell2mat(y)];
start=cell2mat(a);
plot3(start(1,1),start(2,1),start(3,1),'bx',record(1,:),
record(2,:),record(3,:));
view([37.5 30]);
title('Hopfield 神经网络状态空间');
xlabel('a(1)');
ylabel('a(2)');
zlabel('a(3)');

%用多个随机点仿真网络，并绘出相应的轨迹
figure;
axis([-1 1 -1 1 -1 1])';
set(gca,'box','on');
axis manual;
hold on;
view([37.5 30]);
plot3(T(1,:),T(2,:),T(3,:),'r*');
color = 'rgbmy';
for i=1:25
a = {rands(3,1)};
[y,Pf,Af] = sim(net,{1 10},{},a);
record=[cell2mat(a) cell2mat(y)];
start=cell2mat(a);
plot3(start(1,1),start(2,1),start(3,1),'kx',
record(1,:),record(2,:),record(3,:),
color(rem(i,5)+1))
drawnow
end
title('Hopfield 神经网络状态空间');
xlabel('a(1)');
ylabel('a(2)');
zlabel('a(3)');

%指定一些特殊的初始点进行仿真，并绘出相应的轨迹
figure;
axis([-1 1 -1 1 -1 1])';
set(gca,'box','on');
axis manual;
```

```
hold on;
P = [ 1.0   -1.0   -0.5   1.00   1.00   0.0; 0.0    0.0    0.0
0.00   0.00 -0.0;-1.0    1.0    0.5 -1.01 -1.00   0.0];
view([37.5 30]);
color ='rgbmy';
for i=1:6
a = {P(:,i)};
[y,Pf,Af] = sim(net,{1 10},{},a);
record=[cell2mat(a) cell2mat(y)];
start=cell2mat(a);
plot3(start(1,1),start(2,1),start(3,1),'kx',
record(1,:),record(2,:),record(3,:),
color(rem(i,5)+1))
drawnow
end
title('Hopfield 神经网络状态空间');
xlabel('a(1)');
ylabel('a(2)');
zlabel('a(3)');
```

第 7 章　自组织与 LVQ 神经网络理论及实例

自组织神经网络（Self-organizing NNs）是无教师学习网络，它模拟人类根据过去经验自动适应无法预测的环境变化。因为没有教师信号，这类网络通常利用竞争的原则进行网络的学习。

本章主要内容包括：

- 自组织竞争网络
- 自组织特征映射神经网络
- 自适应共振理论（ART）
- CPN 模型

7.1　自组织竞争网络

自组织竞争人工神经网络的形成也是受生物神经系统的启发，之所以称为自组织竞争人工神经网络，是因为它一般是由输入层和竞争层构成的两层网络。本节将详细讲述这种网络结构的基本思想和学习规则。

7.1.1　自组织竞争网络的形成

在生物神经系统中，存在一种“侧抑制”的现象，即一个神经细胞兴奋后，通过它的分支会对周围其他神经细胞产生抑制。这种侧抑制式神经细胞之间出现竞争，虽然开始阶段各个神经细胞都处于程度不同的兴奋状态，由于侧抑制的作用，各细胞之间相互竞争的最终结果是：兴奋作用最强的神经细胞所产生的抑制作用战胜了它周围所有其他细胞的抑制作用而“赢”了，其周围的其他神经细胞则全“输”了。

自组织竞争人工神经网络正是基于上述生物结构和现象形成的。它是一种以无教师示教的方式进行网络训练的，具有自组织功能的神经网络，网络通过自身训练，自动对输入模式进行分类。在网络结构上，自组织竞争人工神经网络一般是由输入层和竞争层构成的两层网络。网络没有隐含层，两层之间各神经元实现双向连接，有时竞争层各神经元之间还存在横向连接。在学习算法上，它模拟生物神经系统依靠神经元之间的兴奋、协调与抑制、竞争的作用来进行信息处理的动力学原理，指导网络的学习与工作。

7.1.2 自组织竞争网络的基本思想

自组织竞争人工神经网络的基本思想是：网络竞争层各神经元竞争对输入模式的相应机会，最后仅一个神经元成为竞争的胜者，并对那些与获胜神经元有关的各连接权朝着更有利于它竞争的方向调整，这样获胜神经元就表示对输入模式的分类。

除了竞争方法外，还有通过抑制手段获胜的方法，即网络竞争层各神经元都能抑制所有其他神经元对输入模式的响应机会，从而使自己成为获胜者。

此外，还有一种侧抑制的方法，即每个神经元只抑制与自己相邻的神经元，而对远离自己的神经元则不抑制。因此，自组织竞争人工神经网络自组织自适应的能力进一步拓宽了神经网络在模式识别、分类方面的应用。

7.1.3 两种联想学习规则

这两种联想学习规则是 Instar 学习规则和 Outstar 学习规则。

1．Instar 学习规则

由 r 个输入构成的格劳斯贝格 Instar 模型如图 7-1 所示。

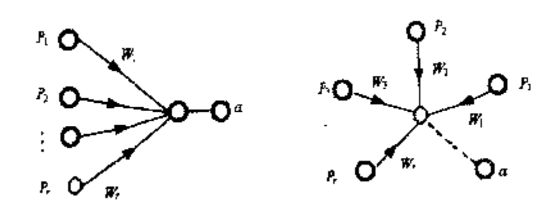


图 7-1　格劳斯贝格 Instar 模型

Instar 模型通过调节网络权值矢量 $\boldsymbol{W}$ 近似于输入矢量 $\boldsymbol{P}$ 来训练某一神经元节点只响应特定的输入矢量 $\boldsymbol{P}$，实现其输入/输出转换的激活函数是硬限制函数。其对权值修正的格劳斯贝格 Instar 学习规则为：

$$\Delta W(i,j) = \eta * (P(j) - W(i,j) * a(i)$$

其中 η 为网络的学习速率。由上式可见，$\Delta W(i,j)$ 与输出成正比。如果格劳斯贝格 Instar 模型的输出矢量 $\boldsymbol{a}$ 被某一外部方式维护高值时，那么通过不断反复地学习，权值将能够逐渐趋近于输入矢量 $P(j)$ 的值，并驱使 $\Delta W(i,j)$ 逐渐减少，直到最终达到 $W(i,j) = P(j)$，从而使 Instar 权矢量学习了输入矢量 P，达到了用 Instar 模型来识别一个矢量的目的。另外，如果 Instar 模型的输出保持为低值时，网络权矢量被学习的可能性较小，甚至不能被学习。

对于一个已训练过的 Instar 模型，当输入端再次出现该学习过的输入矢量时，其产生1 的加权输入和；而与学习过的矢量不相同的输入出现时，所产生的加权输入和总是小于1，由此可见，Instar 加权输入和公式中的权值 W 与输入矢量 P 的点积，反映了输入矢量与网络权矢量之间的相似度，当相似度接近于 1 时，表明输入矢量 P 与权矢量相似，并通

过进一步学习。能够使权矢量对其输入矢量具有更大的相似度。当多个相似输入矢量输入Instar模型时，最终的训练结果是使网络的权矢量趋向于相似输入矢量的平均值。

2. Outstar学习规则

由s个输出节点构成的格劳斯贝格Outstar模型如图7-2所示。

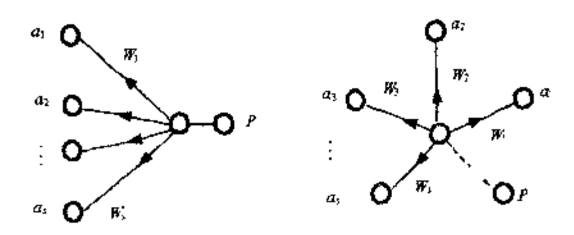


图7-2 格劳斯贝格Outstar模型

Outstar模型被训练来在一层个s线性神经元的输出端产生一个矢量a，其激活函数是线性函数，它被用来学习回忆一个矢量。Outstar模型连接强度的变化ΔW是与输入矢量P成正比的，其对权值修正的格劳斯贝格Outstar学习规则为：

$$\Delta W(i,j)=\eta*(a(j)-W(i,j)*P(i)$$

其中η为网络的学习速率。由上式可见，$\Delta W(i,j)$与输入成正比。如果格劳斯贝格Instar模型的输入矢量$P(j)$被保持高值时，那么通过反复地学习，权值将能够逐渐趋近于输出矢量$a(i)$的值，并驱使$\Delta W(i,j)$逐渐减少，直到最终达到$W(i,j)=a(i0$。当有r个Outstar相并联时，每个Outstar模型与s个线性神经元相连组成一层Outstar时，每当某个Outstar的输入节点被置为1时，与其相连的权值矢量就会被训练成对应的线性神经元的输出矢量。另外，如果Outstar模型的输入保持为低值时，网络的权矢量被学习的可能性较小，甚至不能被学习与修正。

7.1.4 基本竞争型人工神经网络

基本竞争型人工神经网络由输入层和竞争层组成，输入层有N个神经元，竞争层有M个神经元，其网络基本结构如图7-3所示。

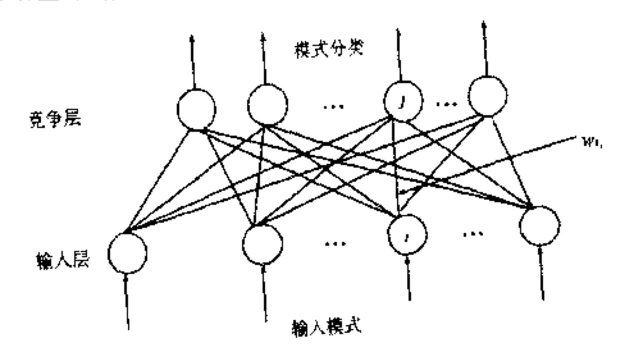


图7-3 基本竞争网络结构

其中网络的连接权为$\{W_{ij}\}$，$i=1,2,...,N$；$j=1,2,...,M$，且约束条件为：

$$\sum_{i=1}^{N}W_{ij}=1 \tag{7.1}$$

网络的T个二值输入学习模式为：$P_k=(p_1^k,p_2^k,...,p_N^k)$，与其对应的竞争层输出模式为：$A_k=(a_1^k,a_2^k,...,a_M^k)$，$k=1,2,...,T$。

网络的学习规则为：

（1）初始化，按照式（7.1）的约束条件赋予$\{W_{ij}\}$为[0,1]区间内的随机值，$i=1,2,...,N$；$j=1,2,...,M$；

（2）任选T个学习模式中的一个模式P_k提供给网络的输入层；

（3）按照下式计算竞争层各神经元的输入值S_j：

$$S_j=\sum_{i=1}^{N}W_{ij}p_i^k,\qquad i=1,2,...,N$$

（4）按照“胜者为王”的原则，以S_j（$j=1,2,...,M$）中最大值所对应的神经元作为胜者，将其输出状态置为1，而其他所有神经元的输出状态值为0，即：

$$\begin{aligned}&a_j=1,\qquad S_j>S_i(i\neq j)\\&a_i=0,\qquad i\neq j\end{aligned} \tag{7.2}$$

如果出现$S_j=S_i$的现象，则按统一约定取左边的神经元为获胜神经元。

（5）与获胜神经元相连的各连接权按照下式进行修正，而其他所有连接权保持不变。

$$w_{ij}=w_{ij}+\Delta w_{ij}$$

$$\Delta w_{ij}=\eta(\frac{p_i^k}{m}-w_{ij})$$

$$i=1,2,...,N(0<\eta<1) \tag{7.3}$$

其中η为学习系数，m为第k个学习模式$P_k=(p_1^k,p_2^k,...,p_N^k)$中元素为1的个数。

（6）选取另一个学习模式，返回步骤（3），直至T个学习模式全部提供给网络。

（7）返回步骤（2），直至各连接权的调整量变得很小为止。

以上的学习规则分析如下：

- 式（7.3）中的学习系数η反映了学习过程中连接权调整量的大小，η的典型值一般为0.01～0.03。

- 由式（7.3）可见，当 P_i 为 1 时，竞争层获胜神经元 j 与输入层神经元 i 之间的连接权 w_{ij} 在满足式（7.1）的约束条件下有 $w_{ij}<1$，所以其调整量为正。即连接权向增大的方向变化；当 P_i 为 0 时，其调整量为负，即连接权向减小的方向变化。所有的连接权始终在（0 1）之间变化。
- 当同一个学习模式反复提供给网络学习后，则这一模式前次所对应的竞争层获胜神经元的输入值 S_j 会逐渐增大，继续保持其胜者的地位。当与这一学习模式非常接近的模式提供给网络时，也将促使同一神经元在竞争中获胜。因此，在网络回想时，就可以根据所记忆的学习模式按照式（7.2）对输入模式做出最邻近分类，即以竞争层获胜神经元表示分类结果。

7.2 自组织特征映射神经网络

自组织特征映射网络（Self-organizing feature map，SOM 网络）是由芬兰赫尔辛基大学神经网络专家 Kohonen 教授在 1981 年提出的，这种网络模拟大脑神经系统自组织特征映射的功能，它是一种竞争式学习网络，在学习中能无监督地进行自组织学习。

自组织特征映射网络应用广泛，可用于语言识别、图像压缩、机器人控制、优化问题等。

7.2.1 自组织特征映射网络的结构

Kohonen 网络结构如图 7-4 所示，它由输入层和竞争层组成。

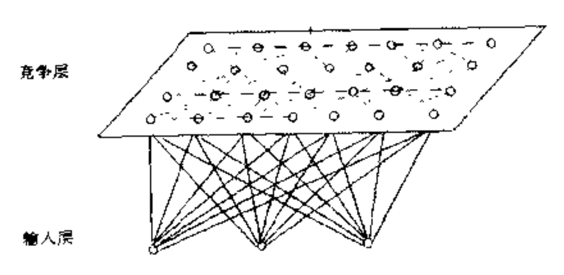


图 7-4 自组织特征映射网络结构

输入层神经元数为 n，竞争层由 $M=m^2$ 个神经元组成，且构成一个二维平面阵列。输入层与竞争层之间实行全互连接，有时竞争层各神经元之间还实行侧抑制连接。网络中有两种连接权，一种是神经元对外部输入反应的连接权值，另一种是神经元之间的连接权值，它的大小控制着神经元之间的交互作用的大小。

7.2.2 自组织特征映射的算法

自组织特征映射算法是一种无教师示教的聚类方法，它能将任意输入模式在输出层映射成一维或二维离散图形，并保持其拓扑结构不变。即在无教师示教的情况下，通过对输入模式的自组织学习，在竞争层将分类结果表示出来。此外，网络通过对输入模式的反复

学习，可以使连接权矢量空间分布密度与输入模式的概率分布趋于一致，即连接权矢量空间分布能反映输入模式的统计特征。

对于输入模式，神经网络的不同区域具有不同的响应特征。通常只有一个神经元或局部区域的神经元对输入模式有积极响应。图 7-5 显示了二维阵列分布的自组织特征映射网络，输入模式 $X=[x_1,x_2,...x_n]^T$ 并行连接到网络的每一个神经元，而每个神经元对应一个权向量 m，它是网络的可调参数。对于输入模式 X，每个神经元的权向量都和其进行比较，距离最近的权向量会自动调节直到与输入模式 X 的某一最大主分量的方向相重合为止。

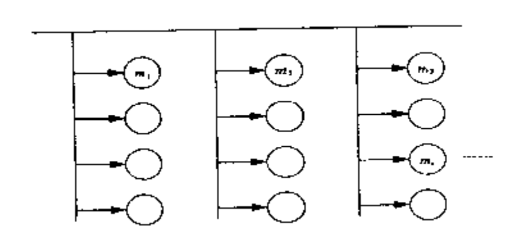


图 7-5　二维阵列的自组织特征映射网络

自组织特征映射的算法，具体过程如下。对于输入模式 X，首先确定中心神经元 M_c，满足 $\|X-M_c\|=\min\{\|X_i-M_i\|\}$。然后，对以 m_c 为中心的周围的神经元的权向量按下式进行调整。

$$M_i(k+1)=\begin{cases}M_i(k)+a(k)[X-M_i(k)],i\in N_c(k)\\M_i(k),i\notin N_c(k)\end{cases}$$

其中 N_c 表示由 m_c 为中心的周围神经元组成的领域。在学习中，$N_c(k)$ 的初始可选大些，然后逐步收缩，通常，学习系数 $a(k)$ 在初始时可取接近于 1.0 的常数，然后逐渐变小，例如 $a(k)$ 可取为 $0.9(1-k/1000)$。

7.2.3　自组织特征映射网络的学习及工作规则

将图 7-5 所示的自组织映射网络结构中各输入神经元与竞争层神经元 j 的连接情况抽出，如图 7-6 所示。

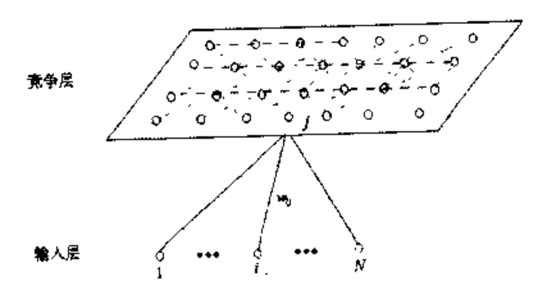


图 7-6　输入神经元与竞争层神经元 j 的连接示意

设网络的输入模式为 $P_k=(p_1^k,p_2^k,...,p_N^k)$， $k=1,2...,q$ 。竞争层神经元矢量为 $A_j=(a_{j1},a_{j2},...,a_{jm}$， $j=1,2,...,m$)。其中，P_k 为连续值，A_j 为数字量。竞争层神经元 j 与输入层神经元之间的连接权矢量为 $W_j=(w_{j1},w_{j2},...,w_{jN}),i=1,2,...,N$； $j=1,2,...,M$ 。

Kohonen 学习规则是由 Instar 规则发展而来的。对于其值为 0 或 1 的 Instar 模型输出，只对输出为 1 的 Instar 权矩阵进行修正，即学习规则只应用于输出为 1 的 Instar 上，将 Instar 模型的学习规则中的输出 a 取值 1，则可导出 Kohonen 规则。Kohonen 网络的自组织学习过程包括两个部分：一是选择最佳匹配神经元，二是权矢量自适应变化的更新过程。

Kohonen 网络的自组织学习过程也可以描述为：对于每一个网络的输入，只调整一部分权值，使权向量更接近或更偏离输入矢量，这一调整过程就是竞争学习。随着不断的学习进程，所有权矢量都在输入矢量空间相互分离，形成了各自代表输入空间的一类模式，这就是 Kohonen 网络的特征自动识别的聚类功能。

网络的学习及工作规则为：

（1）初始化。将网络的连接权 $\{W_{ij}\}$ 赋予[0，1]区间内的随机值，$i=1,2,...,N$；

$j=1,2,...,M$ 。确定学习速率 $\eta(0)$ 的初始值 $\eta(0)(0<\eta(0)<1)$；确定邻域 $N_g(t)$ 的初始值 $N_g(0)$。邻域 $N_g(t)$ 是指以步骤（4）确定的获胜神经元 g 为中心，且包含若干神经元的区域范围。这个区域一般是均匀对称的，最典型的是正方形或圆形区域，如图 7-7 所示。$N_g(t)$ 的值表示在第 t 次学习过程中邻域中所包含的神经元的个数；确定总的学习次数 T 。

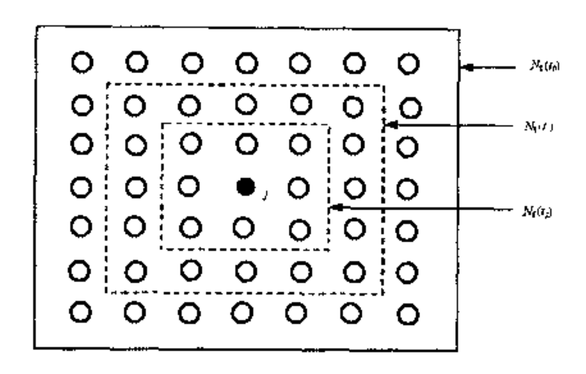


图 7-7　被选神经元 j 及其邻域变化

（2）任选 q 个学习模式中的一个模式 P_k 提供给网络的输入层，并进行归一化处理。

$$\bar{P_k}=\frac{P_k}{\|P_k\|}=\frac{(p_1^k,p_2^k,...,p_n^k)}{[(p_1^k)^2+(p_2^k)^2+...+(p_n^k)^2]^{1/2}}$$

（3）对连接权矢量 $W_j=(w_{j1},w_{j2},...,w_{jN})$ 进行归一化处理，计算 $\bar{W}_j$ 与 $\bar{P}_k$ 之间的欧氏距离：

$$\overline{w_j}=\frac{w_j}{\|w_j\|}=\frac{(w_{j1},w_{j2},...,w_{jn})}{[(w_{j1})^2+(w_{j2})^2+...+(w_{jn})^2]^{1/2}}$$

$$d_j=[\sum_{i=1}^{N}(\overline{p_i^k}-\overline{w_{ji}})^2]^{1/2},\ j=1,2,...,M$$

（4）找出最小距离 d_g，确定获胜神经元 g。

$$d_g=\min[d_j],\qquad j=1,2,...,M$$

（5）进行连接权的调整。对竞争层邻域 $N_g(t)$ 内的所有神经元与输入层神经元之间的连接权进行修正。

$$\overline{w_{ji}(t+1)}=\overline{w_{ji}(t)}+\eta(t).[\overline{p_i^k}-\overline{w_{ji}(t)}]$$

$$j\in N_g(t),\qquad j=1,2,...,M\qquad (0<\eta(t)<1)$$

其中 $\eta(t)$ 为 t 时刻的学习速率。

（6）选取另一个学习模式提供给网络的输入层，返回步骤（3），直至 q 个学习模式全部提供给网络。

（7）更新学习速率 $\eta(t)$ 及邻域 $N_g(t)$。

$$\eta(t)=\eta(0)(1-\frac{t}{T})$$

其中 $\eta(0)$ 为初始学习速率，t 为学习次数，T 为总的学习次数。

设竞争层某神经元 g 在二维阵列中的坐标值为 (x_g,y_g)，则邻域的范围是以点 $(x_g+N_g(t),y_g+N_g(t))$ 和点 $(x_g-N_g(t),y_g-N_g(t))$ 为右三角和左下角的正方形。其修正公式为：

$$N_g(t)=INT[N_g(0)(1-\frac{t}{T})]$$

式中 $INT[x]$ 为取整符号，$N_g(0)$ 为 $N_g(t)$ 的初始值。

（8）令 $t=t+1$，返回步骤（2），直至 $t=T$ 为止。

7.3　自适应共振理论（ART）

对于 BP 网络，如果学习所用的模式是已知，且是固定的，那么网络经过反复学习可以记住这些固定模式。一旦出现新的模式，网络的学习往往会修改甚至删除已学习的结果，从而网络只记得最新的模式。

ART 网络的最初模型为 ART1，只用于二进制输入，后来有可适用于连续信号输入的 ART2，然后又发展了 ART3。

ART 网络是一种向量模式的识别器，它根据存储的模式对输入向量进行分类，其简化的结果如图 7-8 所示。当存储的模式中有的模式和输入模式相匹配时，代表该存储模式的参数就被调整以更接近输入模式。反之，如果在存储模式中，没有发现和输入模式相匹配的模式时，则输入模式作为新的模式被存储到网络中，其他的存储模式保持不变。

如图 7-8 所示，ART 网络由比较和识别的两层神经元组成，增益控制 1、2 和复置用来控制网络的学习和分类。

ART 网络的工作过程如下。当没有输入时，X 的所有成分为 0，使得增益矩阵 2 的信号为 0，因此也使得识别层的输出全为 0。当加入输入向量 X 时，因为 X 必含有不为 0 的元素，因此增益控制 1 和 2 的信号均为 1，从而使比较层的输出向量 C 和输入向量 X 完全相同。接着识别层寻找和 C 最匹配的神经元 j，并使其输出为 1，即 $r_j=1$，其他神经元输出均为 0。然后由 $r_j=1$ 决定比较层中对应的存储模式 $T_j=\{t_{j1},t_{j2},\dots,t_{jm}\}$ 作为向量 P。由于 $r_j=1$，增益控制 1 的信号被强制为 0，这时比较层的输出就是由比较 X 和 P 产生的结果。如果由上而下的反馈信号和输入模式不匹配（P 和 X 对应的成分不同）时，C 的相应成分就变为 0。因此若 C 的成分中 0 多，而 X 的成分中 1 多时，表明由上而下的反馈模式 P 不是所寻找的模式，此时产生复置信号，使识别层被激活的神经元复原，即使其输出为 0。

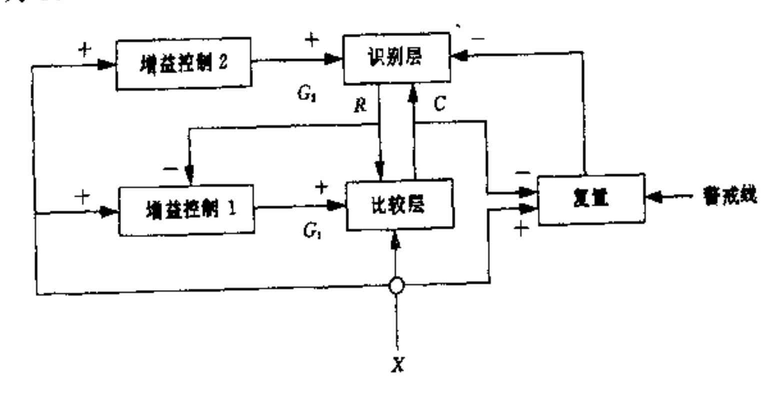


图 7-8　ART 网络的简化结构

如果没有产生复置信号，则表明由上而下的反馈模式和输入模式匹配。反之，则必须搜索其他存储模式，看是否和输入模式相匹配，这一过程反复进行，直到找到一个相匹配的存储模式。如果在所有的存储模式中都没有找到相匹配的模式时，输入模式作为新的模式存储到网络中。

自适应谐振理论（ART）网络具有不同的版本。图 7-9 表示 ART-1 版本，用于处理二元输入。新的版本，如 ART- 2，能够处理连续值输入。

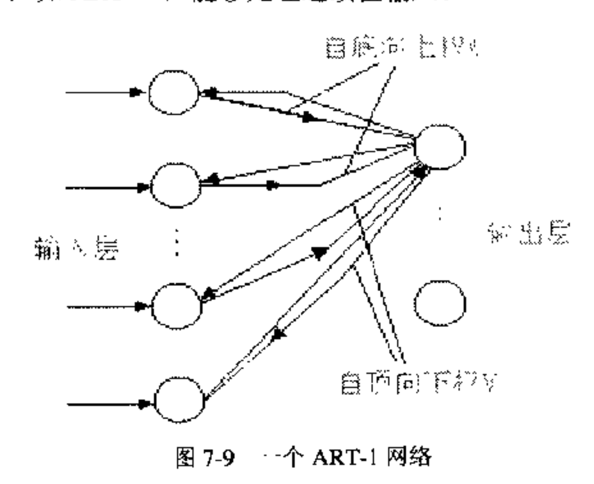


图 7-9 一个 ART-1 网络

从图 7-9 可见，一个 ART-1 网络含有两层，一个输入层和一个输出层。这两层完全互连，该连接沿着正向（自底向上）和反馈（自顶向下）两个方向进行。自底向上连接至一个输出神经元 i 的权矢量 W_i 形成它所表示的类的一个样本。全部权矢量 W_i 构成网络的长期存储器，用于选择优胜的神经元，该神经元的权矢量 W_i 最相似于当前输入模式。自顶向下从一个输出神经元 i 连接的权矢量用于警戒测试，即检验某个输入模式是否足够靠近已存储的样本。警戒矢量 V_i 构成网络的短期存储器。V_i 和 W_i 是相关的，W_i 是 V_i 的一个规格化副本，即：

$$W_i = \frac{V_i}{e + \sum V_{ji}} \tag{7.4}$$

在式（7.4）中，e 为一小的常数，V_{ji} 为 V_i 的第 j 个分量（即从输出神经元 i 到输入神经元 j 连接的权值）。

当 ART-1 网络在工作时，其训练是连续进行的，且包括下列步骤：

（1）对于所有输出神经元，预置样本矢量 W_i 及警戒矢量 V_i 的初值，设定每个 V_i 的所有分量为 1，并据式（7.4）计算 W_i。如果一个输出神经元的全部警戒权值均置 1，则称为独立神经元，因为它不被指定表示任何模式类型。

（2）给出一个新的输入模式 x。

（3）使所有的输出神经元能够参加激发竞争。

（4）从竞争神经元中找到获胜的输出神经元，即这个神经元的 $x.W_i$ 值为最大；在开始训练时或不存在更好的输出神经元时，优胜神经元可能是一个独立神经元。

（5）检查该输入模式 x 是否与获胜神经元的警戒矢量足够相似。相似性是由 x 的微分

式 r 检测的，即：

$$r=\frac{x\cdot V_i}{\sum x_i} \tag{7.5}$$

如果 r 值小于警戒阈值 $r\,(0<r<1)$，那么可以认为 x 与 V_i 是足够相似的。

（6）如果 $r3\ r$，即存在谐振，则转向第（7）步；否则，使获胜神经元暂时无力进一步竞争，并转向第（4）步，重复这一过程直至不存在更多的有能力的神经元为止。

（7）调整最新获胜神经元的警戒矢量 V_i，对它逻辑加上 x，删去 V_i 内而不出现在 x 内的位；据式（7.4），用新的 V_i 计算自底向上样本矢量 W_i；激活该获胜神经元。

（8）转向第（2）步。

上述训练步骤能够做到：如果同样次序的训练模式被重复地送至此网络，那么其长期和短期存储器保持不变，即该网络是稳定的。假定存在足够多的输出神经元来表示所有不同的类，那么新的模式总是能够学得，因为新模式可被指定给独立输出神经元，如果它不与原来存储的样本很好匹配的话（即该网络是塑性的）。

总的说来，ART 的特点如下：

- ART 是一种非监督，向量类聚的竞争学习算法。
- ART 对“弹性—稳定性”问题提供一种解决办法。
- ART 是以认知学和行为学为基础的模型。
- ART 在输入与输出层使用大量反馈连接。
- ART 可用非线性微分方程组描述。
- ART 能工作于实数模式或二值模式输入。

7.4　CPN 模型

CPN（Counter Propagation Network），它是利用了自组织映射近似函数的一种新的映射神经网络，网络结构的主要特点是组合 Kohonen 的自组织映射和 Grossberg 的外星（Outstar）结构。图 7-10 是前向传递的 CPN 模型，前向传递的 CPN 由三层构成：输入层、Kohonen 层和 Grossberg 层。

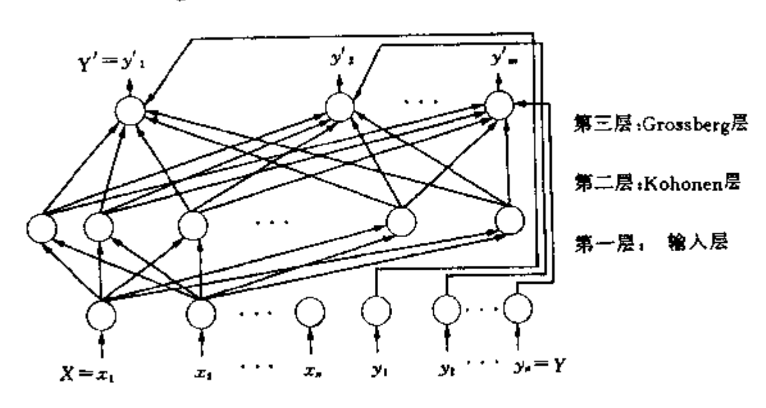


图 7-10　前向传递的 CPN

各层的基本特性和作用可描述如下：

第一层：输入层，提供网络学习用的向量对 (X,Y)，且 $Y=f(X)$

第二层：Kohonen 层，由 N 个神经元组成，其输出分别为 $z_1,z_2,\ldots,Z_N$；且：

$$z_i=\begin{cases}1, & \text{若对所有}j, \|W_i-X\|\leqslant\|W_j-X\| \\ 0, & else\end{cases} \tag{7.6}$$

这里$\|\ .\ \|$为欧基里德距离。对于输入向量 X，Kohonen 层只有与其最近的神经元被激活，输出为 1，其他神经元输出都为 0。此时被激活的神经元的权向量按下式：

$$W_i^{new}=W_i^{old}+a(k)(X-W_i^{old})$$

其中$0\leqslant a(k)\leqslant 1$，起始时取较大的数，如 0.8，然后随着训练逐渐减小。当训练结束后，权向量保持不变，此时只有（7.4）式用来决定中间层的输出。

第三层：Grossberg 层，由 m 个神经元组成，其输出分别为 $y_1^{'},y_2^{'},\ldots,y_m^{'}$，且：

$$y_j^{'}=\sum_{i=1}^{N}t_{ji}^{old}z_i$$

$$t_{ji}^{new}=t_{ji}^{old}+\beta(y_j-y_j^{'})z_i$$

这里 $T_j=(t_{j1},t_{j2},\ldots t_{jN})^T$ 是连接第三层的神经元 j 的权向量，$\beta(0<\beta<1)$ 是 Grossberg 学习规则中的学习常数。由于中间层的输出中只有一个为 1（即 $z_i=1$），因此第三层神经元 j 的输出 $y_j^{'}$ 就取 t_{ji}，这里 t_{ji} 是第二层神经元 i 和第三层神经元相连的权值，且只有权向量 T_j 根据 Grossberg 学习规则得到调节。当训练完成后，网络将输出对应激活第二层同一神经元的向量对 (X,Y) 中所有 Y 的平均值。

原则上讲，增加第二层神经元的数目，可以提高函数近似的精度，因此和 BP 网络一样，CPN 可用来近似一般的连续函数。但实际上，要达到一定的精度，所需的神经元数目比 BP 网络要大得多。尽管如此，CPN 具有潜在的应用前景，可用于图像处理和统计分析。

第 8 章　自组织与 LVQ 神经网络应用设计分析

自组织神经网络是神经网络领域中最吸引人的话题之一。这种结构的网络能够从输入信息中找出规律，以及关系，并且根据这些规律来相应地调整网络，使得以后的输出与之相适应。前面介绍了自组织与 LVQ 神经网络的理论并列举了一些实例，本章将介绍自组织与 LVQ 神经网络的设计方法。

本章主要内容：

- 自组织竞争网络在模式分类中的应用
- 二维自组织特征映射网络设计
- LVQ 模式分类网络设计

8.1　引　　言

自组织神经网络是神经网络领域中最吸引人的话题之一。这种结构的网络能够从输入信息中找出规律，以及关系，并且根据这些规律来相应地调整网络，使得以后的输出与之相适应。

在 MATLAB 6.5 提供的神经网络工具箱中，自组织网络被分为自组织竞争网络和自组织特征映射网络两种。

自组织竞争网络能够识别成组的相似输入向量，常用于进行模式分类。在本章的第 2 节，我们将设计一个自组织竞争神经网络，对输入样本向量进行学习，使其能够对一般的输入向量进行分类。

自组织特征映射神经网络根据输入向量在输入空间的分布情况对它们进行分类。自组织特征映射神经网络不但能够学习输入的分布情况（这一点和自组织竞争神经网络一样），而且可以学习进行训练的输入向量的拓扑结构。在本章的第 3 节，我们将设计一个自组织特征映射网络，对二维的输入向量进行学习，使其能够根据输入向量的分布，相应地在输入空间中表示出不同的区域。

学习向量量化（LVQ）神经网络则是一种有监督的训练竞争层的方法。竞争层自动学习对输入向量的分类。但是，竞争层进行的分类只与输入向量之间的距离有关。如果输入向量非常相近，那么竞争层就很有可能将它们归到一类。在本章的第 4 节，我们将设计一个 LVQ 模式分类神经网络。

8.2 自组织竞争网络在模式分类中的应用

自组织竞争网络的简单工作过程是网络输入模式向量后，按照某一规则让输出层节点开始竞争，当某一节点竞争获胜后，对权结构按照能使获胜的节点对该类模式更加敏感的方向进行调整。当网络再输入这个模式或者相近模式时，该节点更易获胜，同时，其他节点受到抑制，从而对该类模式不敏感而难以获胜。当有其他类模式输入时，这些节点再参与有希望的竞争。

自组织竞争神经网络的这些特性使得其适合在模式分类中应用。下面我们就要使用自组织竞争神经网络设计一个模式分类器。

自组织竞争神经网络的结构如图 8-1 所示。

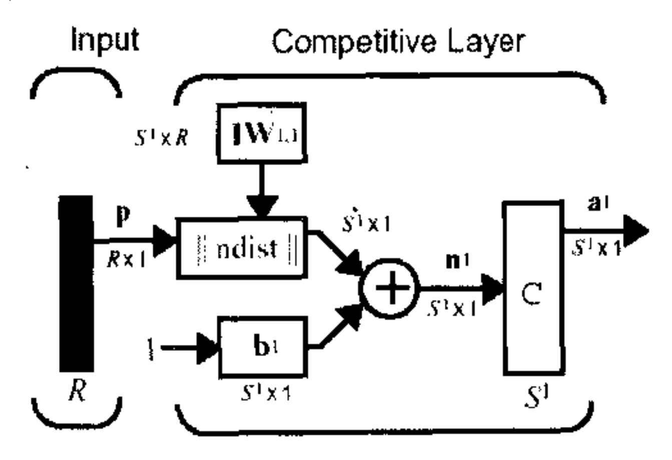


图 8-1　自组织竞争神经网络结构

8.2.1　问题描述

给网络输入一些类别的样本向量，经过训练后，网络调整权值。对于以后输入的向量，网络能够将它们进行正确的分类。

使用 nngenc 函数来产生一定类别的样本向量。nngenc 函数有四个参数，第一个用于指定类中心的范围，第二个用于指定类别的数目，第三个用于指定每一类的样本点的数目，第四个用于指定每一类样本的标准差。

在本问题中，我们所指定类中心范围为 0～1，类别数目为 5，每一个类别有 10 个样本点，每一类样本的标准差为 0.05。

```
X = [0 1; 0 1];     % 限制类中心的范围
clusters = 5;       % 指定类别数目
points = 10;        % 指定每一类的点的数目
std_dev = 0.05;     % 指定每一类的标准差
P = nngenc(X,clusters,points,std_dev);
plot(P(1,:),P(2,:),'+r');
```

```
title('输入向量');
xlabel('p(1)');
ylabel('p(2)');
```

图 8-2 中显示了这些输入样本点的分布情况。

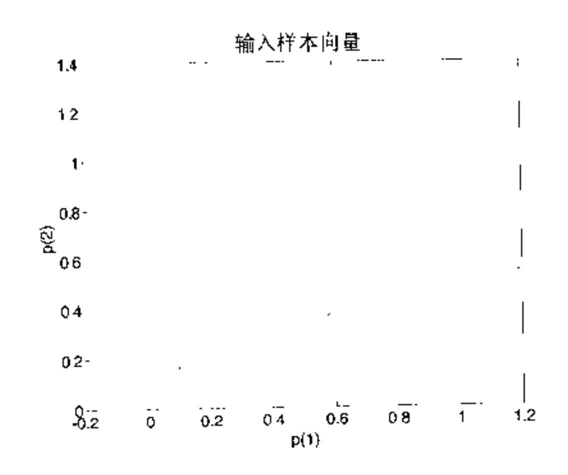


图 8-2　输入样本向量的分布

从图 8-2 中可以看到，这些随机产生的样本向量按我们的设计分成了五类，虽然不是很明显，但还是能够看出来。

网络的工作就是对这些已知的分类样本向量进行学习，调整网络的权值，使得以后对网络输入向量能够进行正确的分类。

8.2.2　网络建立

使用函数 newc 来建立自组织竞争神经网络。由于要区分的类别数目为 5，所以设置网络的神经元数目也为 5。设置学习速率为 0.1。

```
net=newc([0 1;0 1],5,0.1); %设置神经元数目为 5
```

下面求出网络的初始权值，并在图上绘制出。

```
plot(P(1,:),P(2,:),'+r');
w=net.iw{1}
hold on;
plot(w(:,1),w(:,2),'ob');
hold off;
title('输入样本向量及初始权值');
xlabel('p(1)');
ylabel('p(2)');
```

得到网络初始权值：

```
w =
0.5000    0.5000
0.5000    0.5000
0.5000    0.5000
```

```
0.5000    0.5000
0.5000    0.5000
```

图 8-3 在同一幅图中显示了输入样本向量与网络初始权值的分布情况，分别用“+”和“o”标记。

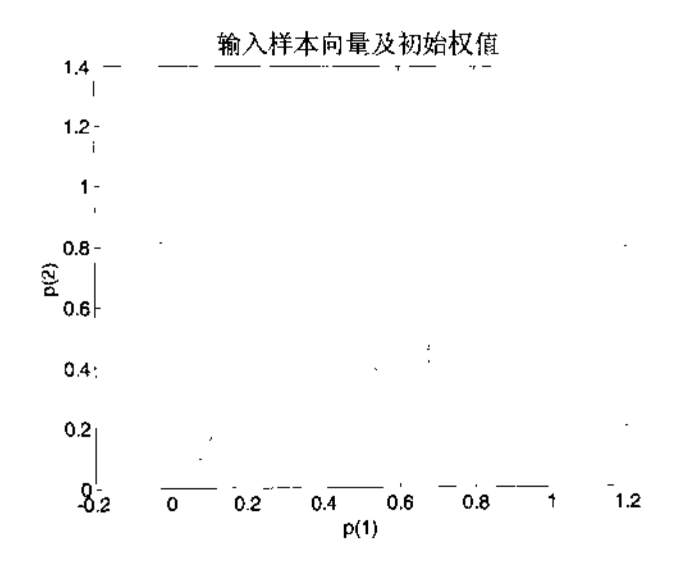


图 8-3 输入样本向量及网络初始权值

从 MATLAB 命令行的输出，以及图 8-3 均可看出，网络在初始时将所有权值均设置为 0.5，这正好是建立网络时设置的输入向量范围的中间值。

8.2.3 网络训练

下面根据样本向量的分类情况调整网络的权值，使用 train 函数来对网络进行训练。设置最大训练步数为 7。

```
net.trainParam.epochs=7;
net=init(net);
net=train(net,P);
```

对于训练好以后的网络，同样可以得到竞争层的权值，并在图 8-4 中显示出它们的分布情况。

```
w=net.iw{1}
plot(w(:,1),w(:,2),'ob');
hold off;
title('输入样本向量及更新后的权值');
xlabel('p(1)');
ylabel('p(2)');
```

训练后的网络竞争层权值为：

```
w =
0.8233    0.0627
0.5501    0.9315
```

```
0.9951      0.5956
0.4071      0.9680
0.4676      0.3557
```

与前面一样，分别用“+”和“o”标记出输入样本向量和训练后的网络竞争层权值分布情况，如图 8-4 所示。

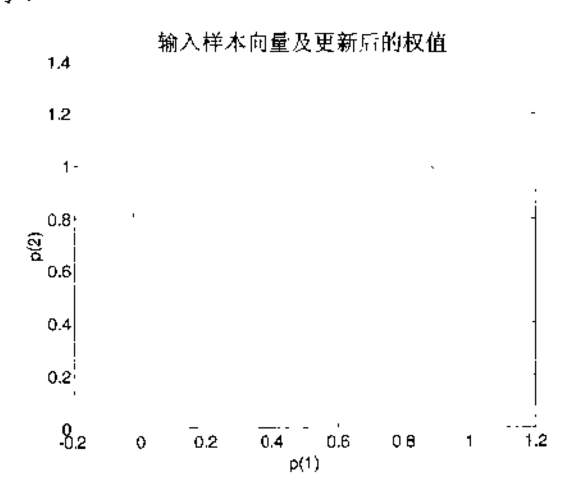


图 8-4　训练后的网络权值分布

从图 8-4 可以看出，经过训练以后，网络的权值得到了调整。调整后的权值分布在各个类的中心位置上。

8.2.4　网络测试与使用

网络训练好以后，其权值就固定下来了。以后对于每一个输入值，网络就会输出相应的分类值。可以利用这一点来进行网络的测试。

实际上，对于训练好的网络，也正是这样使用它的。对于我们需要分类的模式矢量，将其输入到网络中，网络就可以对其分类。

设置一个特定的点，使用 sim 函数来对其进行分类，并在 MATLAB 命令行中得到相应的输出结果。

```
p = [0.6; 0.8];
a=sim(net,p)
```

得到的结果为：

```
a =
(2,1)          1
```

函数 sim 的输出为与竞争层对应的一个稀疏矩阵。这个结果指出了哪一个神经元发生了响应，这也就反映了这个输入属于哪一个类别。

8.2.5　小结

自组织竞争神经网络对给定的输入向量能够进行分类。如果只需要对输入向量进行分

类，那么自组织竞争神经网络就能够很好地完成。自组织竞争神经网络也能够学习输入向量的分布，当然，为了将输入空间进行分类，这就需要更多神经元，特别是当输入向量的密度增大时。自组织特征映射网络在这一点上的优势更多，下一节我们就来设计一个二维的自组织特征映射神经网络。

例程 8-1 是本问题的 MATLAB 程序代码。

例程 8-1

```
%产生指定类别的样本点，并在图中绘制出
    X = [0 1; 0 1];     % 限制类中心的范围
    clusters = 5;       % 指定类别数目
    points = 10;        % 指定每一类的点的数目
    std_dev = 0.05;     % 每一类的标准差
    P = nngenc(X,clusters,points,std_dev);
    plot(P(1,:),P(2,:),'+r');
    title('输入样本向量');
    xlabel('p(1)');
    ylabel('p(2)');
    %建立网络
    net=newc([0 1;0 1],5,0.1); %设置神经元数目为 5
    %得到网络权值，并在图上绘制出
    figure;
    plot(P(1,:),P(2,:),'+r');
    w=net.iw{1}
    hold on;
    plot(w(:,1),w(:,2),'ob');
    hold off;
    title('输入样本向量及初始权值');
    xlabel('p(1)');
    ylabel('p(2)');
    figure;
    plot(P(1,:),P(2,:),'+r');
    hold on;
    %训练网络
    net.trainParam.epochs=7;
    net=init(net);
    net=train(net,P);
    %得到训练后的网络权值，并在图上绘制出
    w=net.iw{1}
    plot(w(:,1),w(:,2),'ob');
    hold off;
    title('输入样本向量及更新后的权值');
    xlabel('p(1)');
    ylabel('p(2)');
    a=0;
```



```
p = [0.6 ;0.8];
a=sim(net,p)
```

8.3　二维自组织特征映射网络设计

自组织特征映射网络根据输入向量在输入空间的分布情况对它们进行分类。与自组织竞争网络不同的是，在自组织映射神经网络中邻近的神经元能够识别输入空间中邻近的部分。这样，自组织特征映射神经网络不但能够学习输入的分布情况（这一点和自组织竞争神经网络一样），而且可以学习进行训练的输入向量的拓扑结构。

自组织特征映射网络结构如图 8-5 所示。

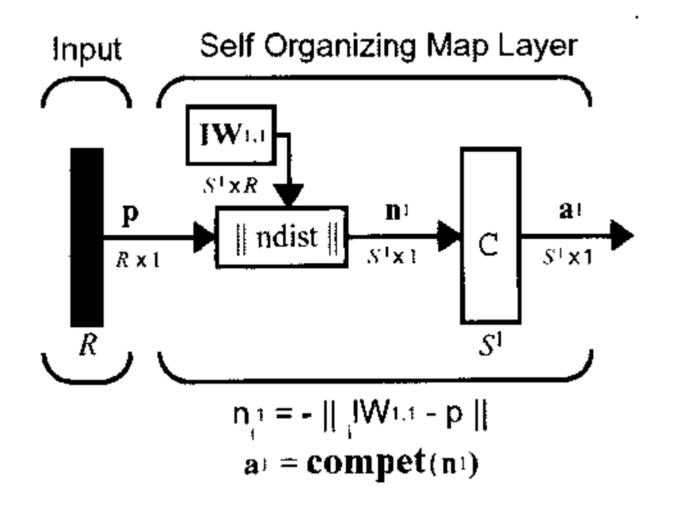


图 8-5　自组织特征映射网络结构

下面设计一个自组织特征映射网络，对二维的输入向量进行学习，使其能够根据输入向量的分布，相应地在输入空间中表示出不同的区域。

8.3.1　问题描述

已知在一个矩形向量区域里，有 1000 个二维向量，这些向量就是网络的输入样本向量。首先使用 rands 函数产生 1000 个这样的二维随机向量。在图 8-6 中显示了这些二维随机向量的分布情况。

```
P=rands(2,1000);
plot(P(1,:),P(2,:),'+r')
title('初始随机样本点分布');
xlabel('P(1)');
ylabel('P(2)');
```

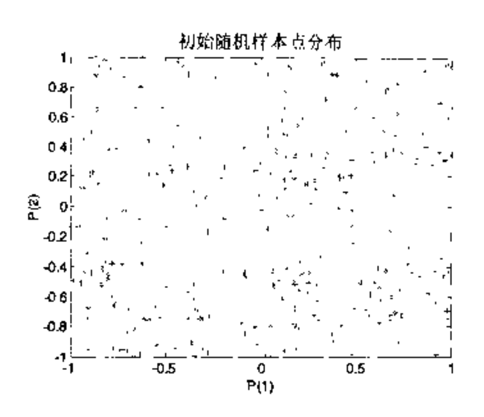


图 8-6　1000 个二维样本向量

8.3.2 网络建立

使用具有 30 个神经元的二维映射网络来对这些输入向量分类。这个二维映射网络的神经元组织结构为 5×6，使用默认的函数来计算距离。我们希望每一个神经元对应矩形中某一个区域产生响应，邻近的神经元对应相邻的区域。使用函数 newsom 来建立自组织特征映射网络。

```
net=newsom([0 1; 0 1],[5 6]);
```

对这个新建立的网络，观察其初始值。

```
net=newsom([0 1; 0 1],[5 6]);
w1_init=net.iw{1,1}
figure;
plotsom(w1_init,net.layers{1}.distances)
```

在 MATLAB 命令行中可以得到初始权值矩阵，由于该矩阵维数太高，限于篇幅，此处不列出。在图 8-7 中，每一个神经元用一个点表示，其坐标值为相应的权值。在初始状态下，这些神经元都拥有相同的权值，即为这些向量的中间值。正因为这样，在图 8-7 中只显示了一个点，实际上是所有点都在这里重合了。

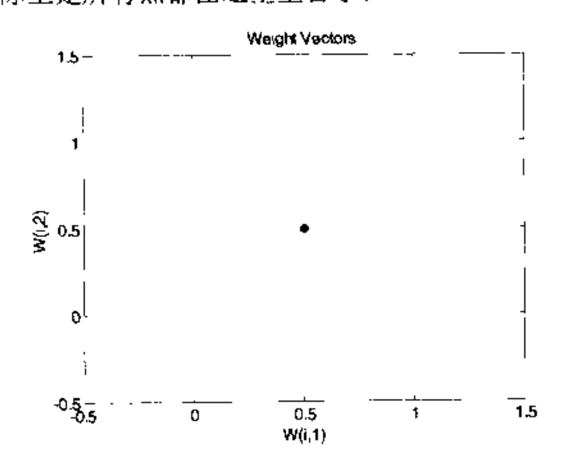


图 8-7　初始权值分布



8.3.3　网络训练

使用函数 train 对建立好的网络进行训练。对 tain 函数只需指定训练步数和样本向量，其余设置使用默认值。由于样本向量有 1000 个，所以训练的过程比较长。分别选择不同的训练步数，观察经过调整得到的相应的权值。

```
for i=10:30:100
net.trainParam.epochs=i;
net=train(net,P);
figure;
plotsom(net.iw{1,1},net.layers{1}.distances)
end
```

在图中绘制出训练后对应每个神经元的权值，这时，由于网络已经进行了训练，就会看到，它们不再像初始时那样重合在一起。

选择训练步数分别为 10、40、70、100，对网络进行训练。相应的权值分布情况分别如图 8-8～图 8-11 所示。

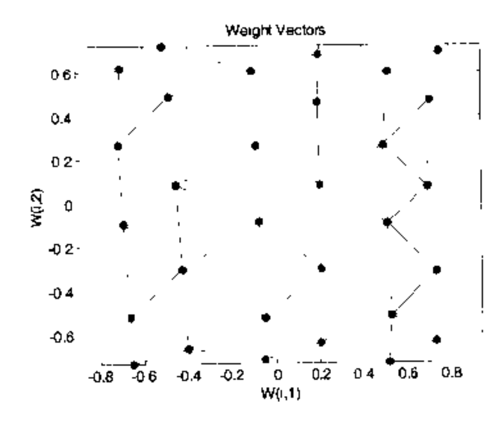


图 8-8　训练步数为 10 时的权值分布

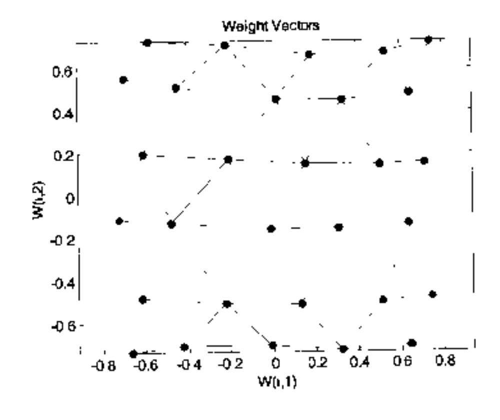


图 8-9　训练步数为 40 时的权值分布

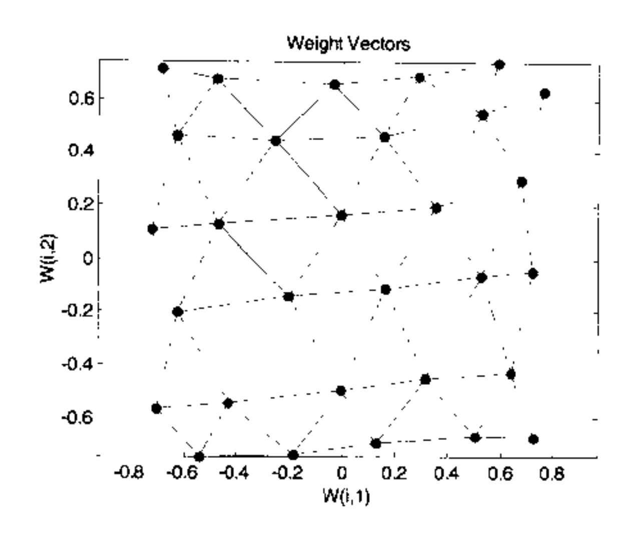


图 8-10　训练步数为 70 时的权值分布

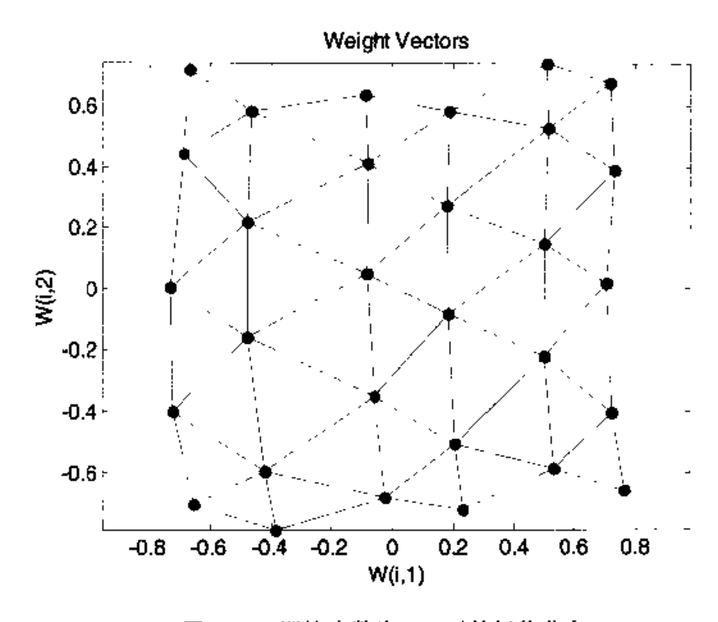


图 8-11　训练步数为 100 时的权值分布

从图 8-8～图 8-11 可以看出，训练 10 步以后，神经元就已经自组织地分布了．每个神经元开始能够区分输入空间中的不同区域。随着训练步数的增加，神经元的权值分布更加合理，但是，当步数达到一定数目以后，这种改变就非常不明显了。如训练 70 步与训练 100 步的效果基本相当。

8.3.4　网络测试与应用

网络训练好以后，其权值就固定下来了。以后对于每一个输入值，网络就会输出相应的分类值。可以利用这一点来进行网络的测试。



实际上，对于训练好的网络，也正是这样使用它的。对于需要分类的模式矢量，将其输入到网络中，网络就可以对其分类。

设置一个特定的点，使用 sim 函数来对其进行分类，并在 MATLAB 命令行中得到相应的输出结果。

```
p = [0.5; 0.3];
a=sim(net,p)
```

得到的结果为：

```
a =
(22,1)          1
```

例程 8-2 是本问题的 MATLAB 程序代码。

例程 8-2

```
%随机生成 1000 个二维向量，作为样本，并绘制出其分布
P = rands(2,1000);
plot(P(1,:),P(2,:),'+r')
title('初始随机样本点分布');
xlabel('P(1)');
ylabel('P(2)');
%建立网络，得到初始权值
net=newsom([0 1; 0 1],[5 6]);
w1_init=net.iw{1,1}
%绘制出初始权值分布图
figure;
plotsom(w1_init,net.layers{1}.distances)
%分别对不同的步长，训练网络，绘制出相应的权值分布图
for i=10:30:100
net.trainParam.epochs=i;
net=train(net,P);
figure;
plotsom(net.iw{1,1},net.layers{1}.distances)
end
%对于训练好的网络，选择特定的输入向量，得到网络的输出结果
p=[0.5;0.3];
a=0;
a = sim(net,p)
```

8.4　LVQ 模式分类网络设计

学习向量量化网络由两层组成，第一层为竞争层，第二层为线性层。竞争层能够学习对输入向量的分类，这与前面的自组织竞争网络非常相似。线性层将竞争层传来的分类信息转变成使用者所定义的类别。将竞争层学习得到的类称为子类，将线性层学习得到的类称为目标类。

下面设计一个学习向量量化网络，根据给定的目标，将输入向量进行模式分类。

8.4.1 问题描述

设输入为二维向量，一共有 10 个。

```
P=[-3 -2 -2  0  0  0  0 +2 +2 +3;
0 +1 -1 +2 +1 -1 -2 +1 -1  0];
```

这些向量属于的类别定义为：

```
C = [1 1 1 2 2 2 2 1 1 1];
```

将这些类别转换成学习向量量化网络使用的目标向量。

```
T = ind2vec(C)
```

用不同的颜色，绘制出这些输入向量。

```
plotvec(P,C),
title('输入二维向量');
xlabel('P(1)');
ylabel('P(2)');
```

结果如图 8-12 所示。

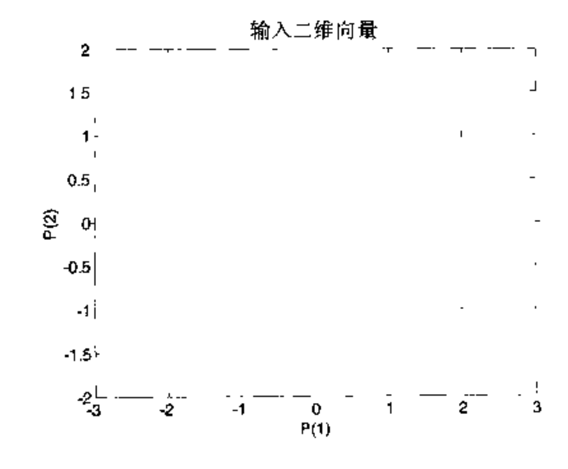


图 8-12 输入二维向量

函数 plotvec 能够根据其第二个参数的情况用不同的颜色在同一幅图上绘制出第一个参数中相应的向量。这里使用默认的标记“+”。（由于印刷关系，此处颜色可能不能够准确表示，但读者在 MATLAB 中可以得到真实颜色的图像。）

8.4.2 网络建立

下面使用函数 newlvq 来建立一个学习向量量化神经网络。

```
net = newlvq(minmax(P),4,[0.6 0.4],0.1);
```

在这里，设置了四个参数。第一个参数表示的是输入向量的范围；第二个参数指定了网络隐含层的神经元的数目，在这里设置为 4；第三个参数设置为[0.6 0.4]，这意味着在第二层的权重中，有 60%的列第一行的值为 1，40%的列第二行的值为 1，即 60%的列属于第一类，40%的列属于第二类；第四个参数指定了学习速率为 0.1。学习函数使用默认值，

为 learnlv1。

绘制出刚建立的网络的权重向量，与输入向量在同一幅图中绘制出。分别用标记“+”与“o”表示输入向量与权重向量，如图 8-13 所示。

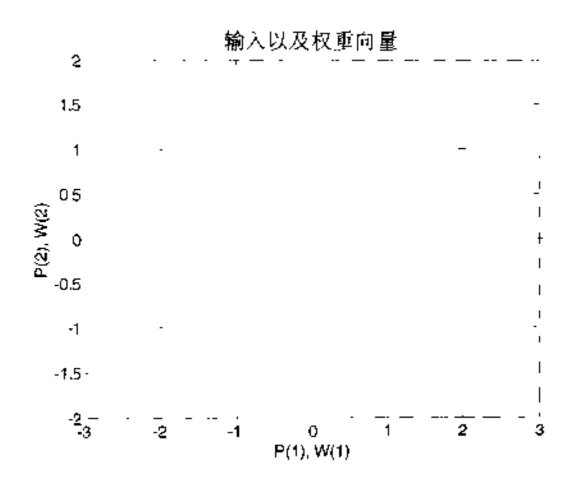


图 8-13 输入向量及初始权重向量

```
plotvec(P,C)
hold on
W1=net.iw{1};
plot(W1(1,1),W1(1,2),'ow')
title('输入以及权重向量');
xlabel('P(1), W(1)');
ylabel('P(2), W(2)');
hold off;
```

可以看出，由于为经过训练，网络的初始权重都相同，为一个中间值，故在图 8-13 中重合到了一个点上。

8.4.3 网络训练

使用函数 train 训练网络。设置训练步数为 150。

```
net.trainParam.epochs=150;
net.trainParam.show=Inf;
net=train(net,P,T);
```

再次在图上绘制出输入向量和训练后得到的竞争层神经元的权重向量，分别用标记“+”和“o”表示。

```
plotvec(P,C);
hold on;
plotvec(net.iw{1}',vec2ind(net.lw{2}),'o');
```

结果如图 8-14 所示。

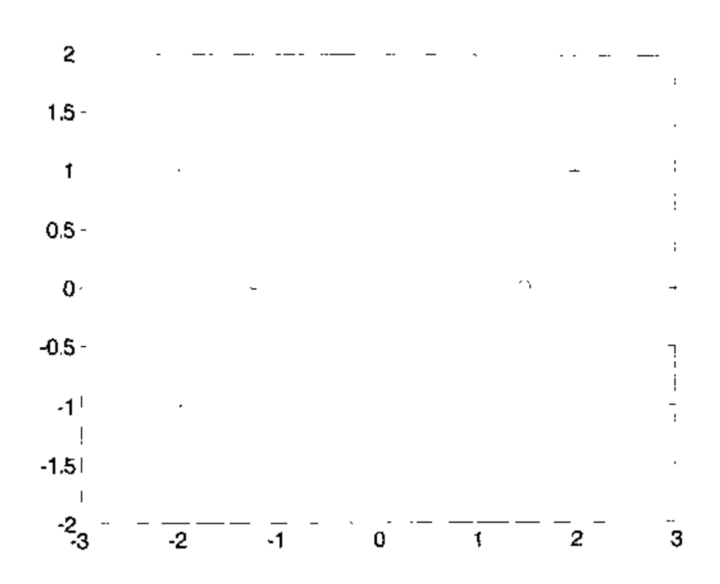


图 8-14　输入向量及训练后的权重向量

从图 8-14 可以看出，经过训练后，竞争层神经元的权重向量分布发生了变化。这种分布显然是适合对两类模式进行分类的。

8.4.4　网络测试及使用

网络训练好以后，其权值就固定下来了。以后对于每一个输入值，网络就会输出相应的分类值。可以利用这一点来进行网络的测试。

设置一个特定的点，使用 sim 函数来对其进行分类，并在 MATLAB 命令行中得到相应的输出结果。

```
p = [0.8; 0.3];
a = vec2ind(sim(net,p))
```

得到结果为：

```
a =
1
```

这说明向量 $\boldsymbol{p}$ 属于第 *1* 类，这与事实是相吻合的。

实际上，对于训练好的网络，也正是这样使用它的。对于我们需要分类的模式矢量，将其输入到网络中，网络就可以对其分类。

8.4.5　结论

学习向量量化网络能够对任意输入向量进行分类，不管它们是不是线性可分的，这一点比感知器神经网络要优越得多。其需要注意的地方是，在竞争层必须要有足够多的神经元，是使得每一个类有足够多的竞争神经元。

例程 8-3 是本问题的 MATLAB 程序代码。

例程 8-3

```
%指定输入二维向量及其类别
P = [-3 -2 -2  0  0  0  0 +2 +2 +3;
0 +1 -1 +2 +1 -1 -2 +1 -1  0];
C = [1 1 1 2 2 2 2 1 1 1];
%将这些类别转换成学习向量量化网络使用的目标向量
T = ind2vec(C)
%用不同的颜色，绘制出这些输入向量
plotvec(P,C),
title('输入二维向量');
xlabel('P(1)');
ylabel('P(2)');
%建立网络
net = newlvq(minmax(P),4,[.6 .4],0.1);
%在同一幅图上绘制出输入向量及初始权重向量
figure;
plotvec(P,C)
hold on
W1=net.iw{1};
plot(W1(1,1),W1(1,2),'ow')
title('输入以及权重向量');
xlabel('P(1), W(1)');
ylabel('P(2), W(2)');
hold off;
%训练网络，并再次绘制出权重向量
figure;
plotvec(P,C);
hold on;
net.trainParam.epochs=150;
net.trainParam.show=Inf;
net=train(net,P,T);
plotvec(net.iw{1}',vec2ind(net.lw{2}),'o');
%对于一个特定的点，得到网络的输出
p = [0.8; 0.3];
a = vec2ind(sim(net,p))
```



第 9 章　神经控制器结构分析

由于分类方法的不同，神经控制器的结构也就有所不同。本章将简要介绍神经控制结构的典型方案，包括NN学习控制、NN直接逆控制、NN自适应控制、NN内模控制、NN预测控制、NN最优决策控制、NN再励控制、CMAC控制、分级NN控制和多层NN控制等。

本章主要内容包括：

- NN 学习控制
- NN 直接逆模型控制
- NN 自适应控制
- NN 内模控制
- NN 预测控制
- NN 自适应判断控制
- 基于 CMAC 的控制
- 多层 NN 控制
- 分级 NN 控制

9.1　NN 学习控制

由于受控系统的动态特性是未知的或者仅有部分是已知的，因此需要寻找某些支配系统动作和行为的规律，使得系统能被有效地控制。在有些情况下，可能需要设计一种能够模仿人类作用的自动控制器。基于规则的专家控制和模糊控制是实现这类控制的两种方法，而神经网络（NN）控制是另一种方法，称它为基于神经网络的学习控制、监督式神经控制或NN监督式控制。图9-1给出一个NN学习控制的结构。

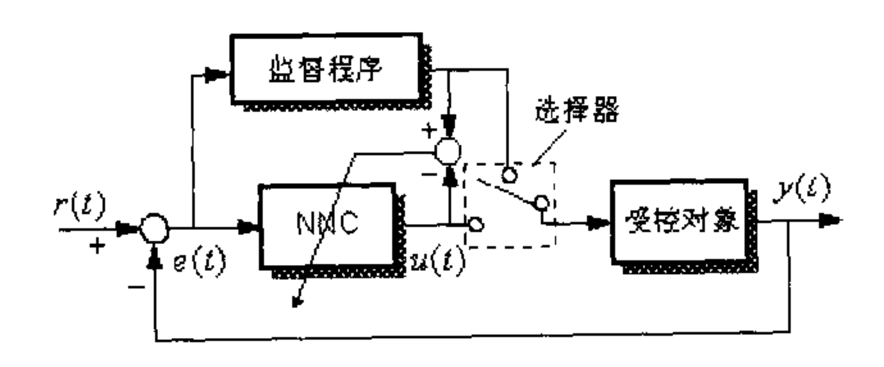


图 9-1　基于神经网络的监督式控制

在图 9-1 中，包括一个导师（监督程序）和一个可训练的神经网络控制器（NNC）。控制器的输入对应于由人接收（收集）的传感输入信息，而用于训练的输出对应于人对系统的控制输入。

实现 NN 监督式控制的步骤如下：

（1）通过传感器和传感信息处理，调用必要的和有用的控制信息。

（2）构造神经网络，选择 NN 类型、结构参数和学习算法等。

（3）训练 NN 控制器，实现输入和输出间的映射，以便进行正确的控制。在训练过程中，可采用线性律、反馈线性化或解耦变换的非线性反馈作为导师（监督程序）来训练 NN 控制器。

NN 监督式控制已被用于标准的倒摆小车控制系统。

9.2 NN 直接逆模型控制

顾名思义，NN 直接逆控制采用受控系统的一个逆模型，它与受控系统串接以便使系统在期望响应（网络输入）与受控系统输出间得到一个相同的映射。因此，该网络（NN）直接作为前馈控制器，而且受控系统的输出等于期望输出。本控制方案已用于机器人控制，即在 Miller 开发的 CMAC 网络中应用直接逆控制来提高 PUMA 机器人操作手（机械手）的跟踪精度。这种方法在很大程度上依赖于作为控制器的逆模型的精确程度。由于不存在反馈，因此本方法的鲁棒性不足。逆模型参数可通过在线学习调整，以期把受控系统的鲁棒性提高至一定程度。

图 9-2 给出 NN 直接逆控制的两种结构方案。在图 9-2（a）中，网络 NN1 和 NN2 具有相同的逆模型网络结构，而且采用同样的学习算法。图 9-2（b）为 NN 直接逆控制的另一种结构方案，图中采用一个评价函数（EF）。

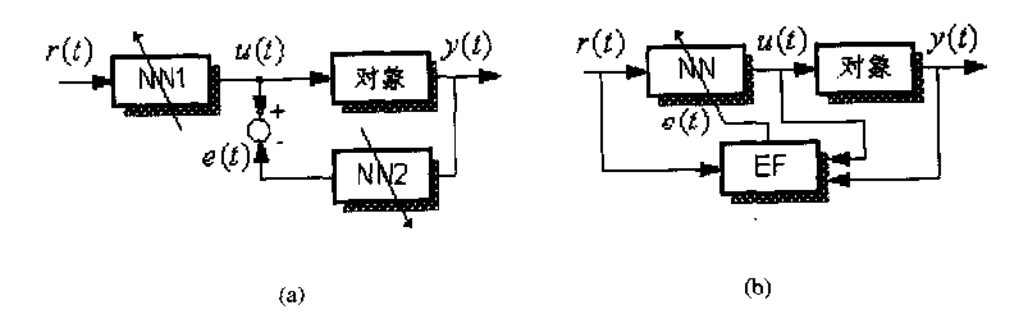


图 9-2　NN 直接逆控制

9.3 NN 自适应控制

与常规自适应控制一样，NN 自适应控制也分为两类，即自校正控制（STC）和模型

参考自适应控制（MRAC）。STC 和 MRAC 之间的差别在于：STC 根据受控系统的正和或逆模型辨识结果直接调节控制器的内部参数，以期能够满足系统的给定性能指标；在 MRAC 中，闭环控制系统的期望性能是由一个稳定的参考模型描述的，而该模型又是由输入一输出对$\{r(t),y^T(t)\}$确定的。本控制系统的目标在于使受控装置的输入 y(t)与参考模型的输出渐近地匹配，即：

$$\lim\left\|y^T(t)-y(t)\right\|\leqslant e$$

式中，e 为一指定常数。

9.3.1 NN 自校正控制（STC）

基于 NN 的 STC 有两种类型，直接 STC 和间接 STC。

1．NN 直接自校正控制

该控制系统由一个常规控制器和一个具有离线辨识能力的识别器组成；后者具有很高的建模精度。NN 直接自校正控制的结构基于上与直接逆控制相同。

2．NN 间接自校正控制

本控制系统由一个 NN 控制器和一个能够在线修正的 NN 识别器组成；图 9-3 表示出 NN 间接 STC 的结构。

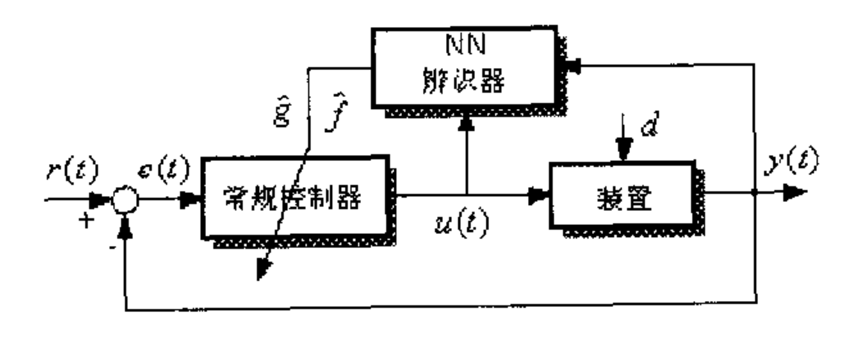


图 9-3 NN 间接自校正控制

一般，我们假设受控对象（装置）为如下式所示的单变量非线性系统：

$$y_{k+1}=f(y_k)+g(y_k)y_k$$

在上式中，$f(y_k)$和$g(y_k)$为非线性函数。令$\hat{f}(y_k)$和$\hat{g}(y_k)$分别代表$f(y_k)$和$g(y_k)$的估计值。如果$f(y_k)$和$g(y_k)$是由神经网络离线辨识的，那么能够得到足够近似精度的$\hat{f}(y_k)$和$\hat{g}(y_k)$，而且可以直接给出常规控制律：

$$u_k=[y_{d,k+1}-\hat{f}(y_k)]/\hat{g}(y_k)$$

式中，$y_{d,k+1}$ 为在（ $k+1$）时刻的期望输出。

9.3.2 NN 模型参考自适应控制

基于 NN 的 MRAC 也分为两类，即 NN 直接 MARC 和 NN 间接 MRAC。

1．NN 直接模型参考自适应控制

从图 9-4 所示的结构可知，直接 MRAC 神经网络控制器力图维持受对象输出与参考模型输出间的差 $e_c(t)=y(t)-y^m(t)\to\infty$ 。由于反向传播需要知道受控对象的数学模型，因而该 NN 控制器的学习与修正已遇到许多问题。

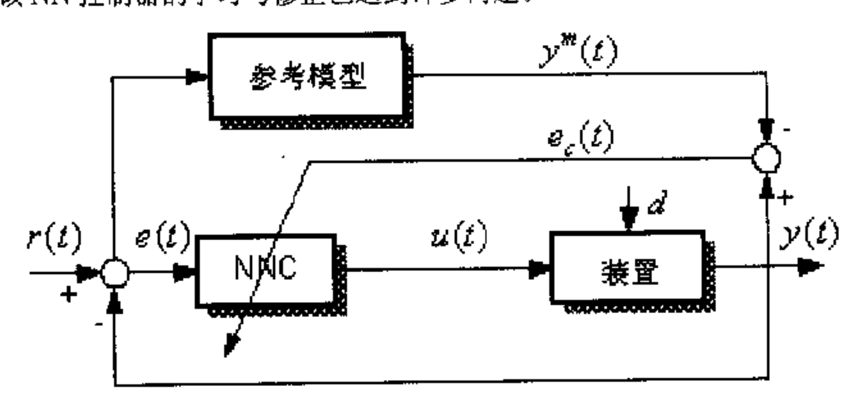


图 9-4 NN 直接模型参考自适应控制

2．NN 间接模型参考自适应控制

该控制系统结构如图 9-5 所示，在图中，NN 识别器（NNI）首先离线辨识受控对象的前馈模型，然后由 $e_i(t)$进行在线学习与修正。显然，NN1 能提供误差 $e_c(t)$或者其变化率的反向传播。

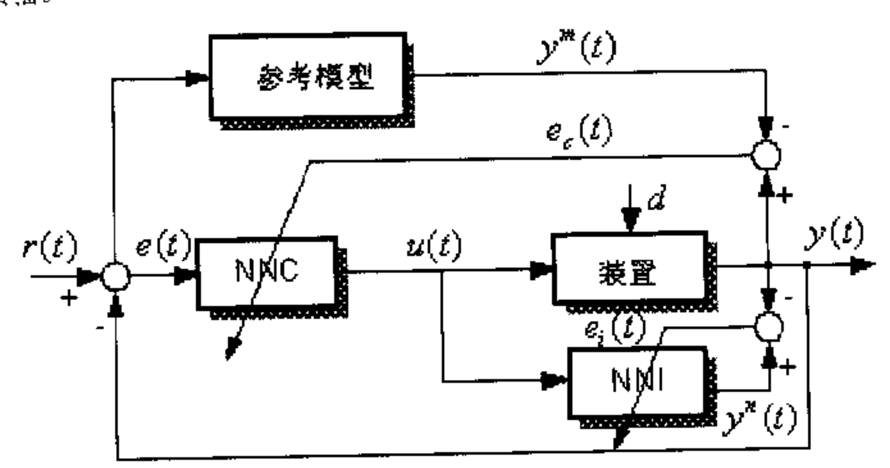


图 9-5 NN 间接模型参考自适应控制



9.4 NN内模控制

在常规内模控制（IMC）中，受控系统的正和逆模型被用做反馈回路内的单元。IMC经全面检验，表明其不仅可用于鲁棒性和稳定性分析，而且是一种新的和重要的非线性系统控制方法。基于NN的内模控制的结构图示于图9-6，其中，系统模型(NN2)与实际系统并行设置。反馈信号由系统输出与模型输出间的差得到，而且由NN1（在正向控制通道上一个具有逆模型的NN控制器）进行处理；NN1控制器应当与系统的逆有关。

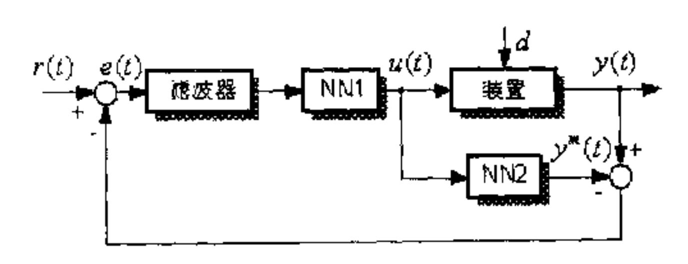


图9-6 NN内模控制

在图9-6中，NN2也是基于神经网络的但具有系统的正向模型。该图中的滤波器通常为一个线性滤波器，而且可被设计满足必要的鲁棒性和闭环系统跟踪响应。

9.5 NN预测控制

预测控制是一种基于模型的控制，它是20世纪70年代发展起来的一种新的控制算法，具有预测模型、滚动优化和反馈校正等特点。已经证明本控制方法对于非线性系统能够产生有希望的稳定性。图9-7表示NN预测控制的一种结构方案。

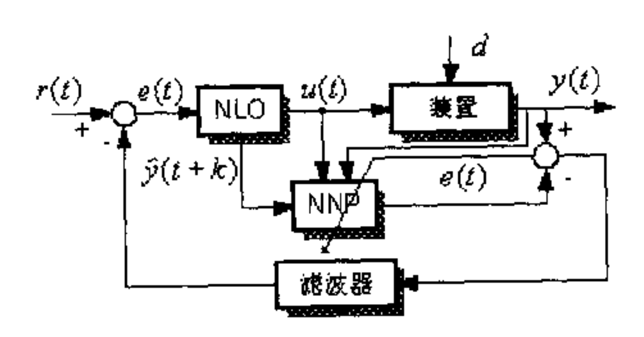


图9-7 NN预测控制

在图9-7中，神经网络预测器NNP为一个神经网络模型，NLO为一个非线性优化器。NNP预测受控对象在一定范围内的未来响应：

$$y(t+j|t), j=N_1, N_1+1, \ldots, N_2$$

式中，N_1和N_2分别叫做输出预测的最小和最大级别，是规定跟踪误差和控制增量的

常数。

如果在时刻（$t+j$）的预测误差定义为：

$$e(t+j)=r(t+j)-y(t+j|t)$$

那么非线性优化器 NLO 将选择控制信号 u(t)使二次性能判据 J 为最小：

$$J=\sum_{j=N_1}^{N_2}e^2(t+j)+\sum_{j=1}^{N_2}\lambda_j\Delta^2u(t+j-1)$$

式中，$Du(t+j-1)=u(t+j-1)-u(t+j-2)$，而 l 为控制权值。

基于神经网络的预测控制算法步骤如下：

（1）计算期望的未来输出序列。

$r(t+j), j=N_1, N_1+1, \ldots, N_2$

（2）借助 NN 预测模型，产生预测输出。

$y(t+j|t), j=N_1, N_1+1, \ldots, N_2$

（3）计算预测误差。

$e(t+j)=r(t+j)-y(t+j|t), j=N_1, N_1+1, \ldots, N_2$

（4）求性能判据 J 的最小值，获得最优控制序列。

u (t+j), j=0, 1, 2, ... N;

（5）采用 u (t)作为第一个控制信号，然后转至第(1)步。

值得说明的是，NLO 实际上为一种最优算法，因此，可用动态反馈网络来代替由本算法实现的 NLO 和由前馈神经网络构成的 NNP。

9.6 NN 自适应判断控制

无论采用何种神经网络控制结构，所有控制方法都有一个共同点，即必须提供受控对象的期望输入。但是，当系统模型未知或部分未知时，就很难提供这种期望输入。NN 自适应判断控制或强化控制是由 Barto 等提出，并由 Anderson 发展的，它应用强化学习的机理。这种控制系统通常由两个网络组成，即自适应判断网络 AJN 和控制选择网络 CSN，如图 9-8 所示。

在本控制系统中，AJN 相当于强化学习需要的“教师”，它起到两种作用：（1）通过不断的奖罚强化学习，使 AJN 逐渐训练为一个熟练的教师；（2）经过学习后，根据受控系统的当前状态和外部强化反馈信号 r(t)，AJN 产生一个强化信号，然后提供内部强化信号，以便能够判断当前控制作用的效果。CSN 相当于多层前馈神经网络控制器，它在内部强化信号的引导下进行学习。通过学习，CSN 根据系统编码后的状态，选择下一个控制作用。

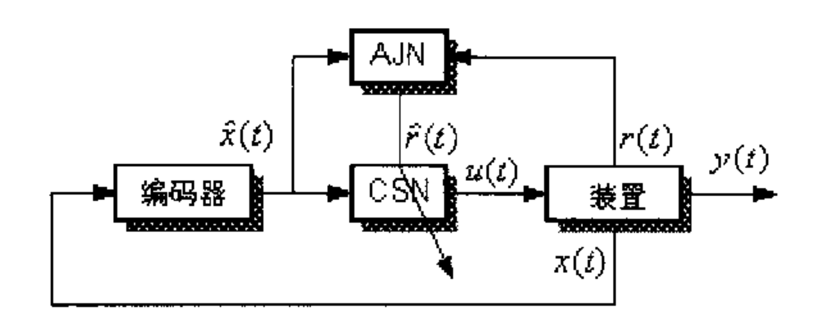


图 9-8　NN 自适应判断控制

9.7　基于 CMAC 的控制

CMAC 是由 Albus 开发的，是近年来获得应用的几种主要神经控制器之一，把 CMAC 用于控制有两种方案。第一种方案的结构如图 9-9 所示。在该控制系统中，指令信号反馈信号均用做 CMAC 控制器的输入。控制器输出直接送至受控装置（对象），必须提供神经网络控制器的期望输出。控制器的训练是以期望输出和控制器实际输出间的差别为基础的。系统工作分两阶段进行。第一阶段为训练控制器，当 CMAC 接收到指令和反馈信号时，它产生一个输出，此输出与期望输出进行比较；如果两者存在差别，那么调整权值以消除该差别。经过这一阶段的竞争，CMAC 已经学会如何根据给定指令和所测反馈信号产生合适的输出，用于控制受控对象。第二阶段为控制，当需要的控制接近所训练的控制要求时，CMAC 就能够很好地工作。这两个阶段工作的完成都无需分析装置的动力学和求解复杂的方程式。不过，在训练阶段，本方案要求期望的装置输入是已知的。

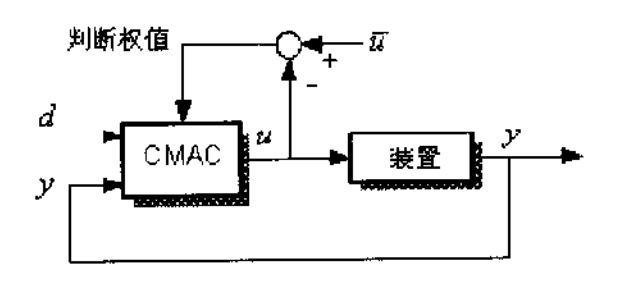


图 9-9　基于 CMAC 的控制（I）

第二种控制方案图示如图 9-10 所示。在本方案中，参考输出方块在每个控制周期产生一个期望输出。该期望输出被送至 CMAC 模块，提供一个信号作为对固定增益常规偏差反馈控制器控制信号的补充。在每个控制周期之末，执行一步训练。在前一个控制周期观测到的装置输出用做 CMAC 模块的输入。用计算的装置输入与实际输入 u 之间的差来计算权值判断。当 CMAC 跟随连续控制周期不断训练时，CMAC 函数在特定的输入空间域内形成一个近似的装置逆传递函数。如果未来的期望输出在域内相似于前面预测的输

出．那么，CMAC 的输出也会与所需的装置实际输入相似。由于上述结果，输出误差将很小，而且 CMAC 将接替固定增益常规控制器。

根据上述说明，方案 I 为一个闭环控制系统，因为除了指令变量外，反馈变量也用做 CMAC 模块的输入，加以编码，使得装置输出的任何变化都能够引起装置接收到的输入的变化。方案 I 中权值判断是以控制器期望输出与控制器实际输出间的误差（而不是装置的期望输出与装置的实际输出间的误差）为基础的。如已述及，这就要求设计者指定期望的控制器输出，并将出现问题，因为设计者通常只知道期望的装置输出。方案 I 中的训练可看做是对一个适当的反馈控制器的辨识。在方案Ⅱ中，借助于常规固定增益反馈控制器，CMAC 模块用于学习逆传递函数。经训练后，CMAC 成为主控制器。本方案中，控制与学习同步进行。本控制方案的缺点是需要为受控装置设计一个固定增益控制器。

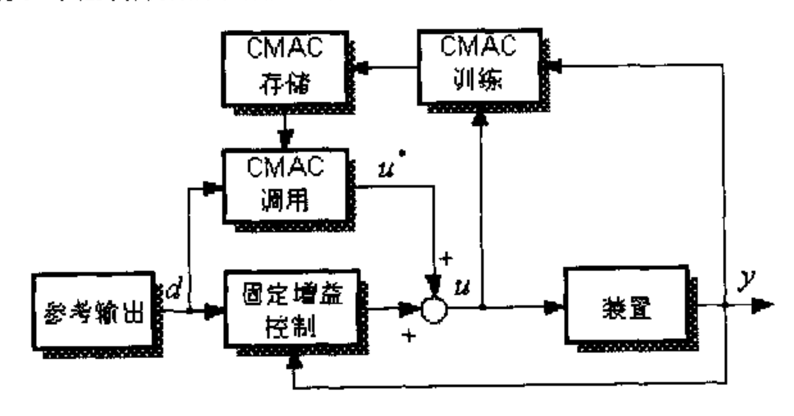


图 9-10 基于 CAMC 的控制（II）

9.8 多层 NN 控制

多层神经网络控制器基本上是一种前馈控制器。让我们考虑图 9-11 所示的一个普通的多层神经控制系统。

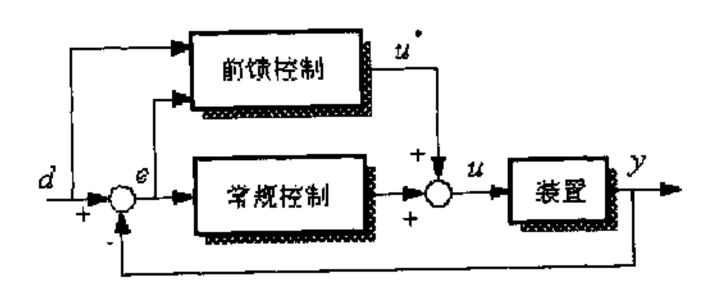


图 9-11 多层 NN 控制的一般结构

该系统存在两个控制作用：前馈控制和常规馈控制。前馈控制由神经网络实现；前馈部分的训练目标在于使期望输出与实际装置输出间的偏差为最小。该误差作为反馈控制器

的输入。反馈作用与前馈作用被分别考虑，特别关注前馈控制器的训练而不考虑反馈控制的存在。已提出多层 NN 控制器的三种结构：间接结构、通用结构和专用结构。

1. 间接学习结构

图 9-12 所示的间接多层 NN 控制结构含有两个同样的神经网络，用于训练。在本结构中，每个网络作为一个逆动态辨识器。训练的目标是要从期望响应 d 中找到一个合适的装置控制 u。以网络 I 和网络 II 间的差为基础来调整权值，使得误差 e 为最小；如果能够训练网络 I 使得 $y=d$，那么，$u=u^*$。不过，这并不能保证期望输出 d 与实际输出 y 之间的差别为最小。

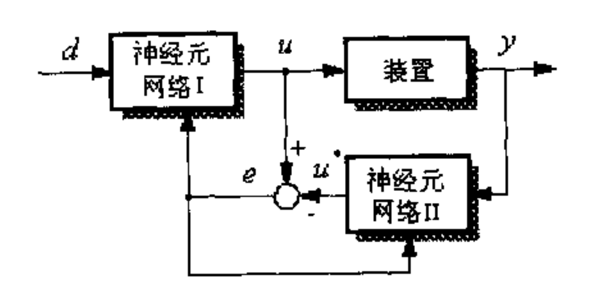


图 9-12　间接学习结构的多层 NN 控制

2. 通用学习结构

图 9-13 绘出多层 NN 控制的通用学习结构，它使图中的 $e=d-g$ 为最小。该网络被训练使得装置输入 u 与网络输出 u^* 间的差别为最小。在训练时，u 应当处在这样的范围内使得 y 复盖期望输出 d。训练之后，如果某一期望输出 d 被送至网络，那么该网络就能够为受控装置提供一个合适的 u。本结构的局限性是：一般无法知道哪一个 u 对应于期望输出 d，因而网络不得不在 u 的大范围内进行训练，以求经过学习能够使装置输出 y 包括期望值 d 。

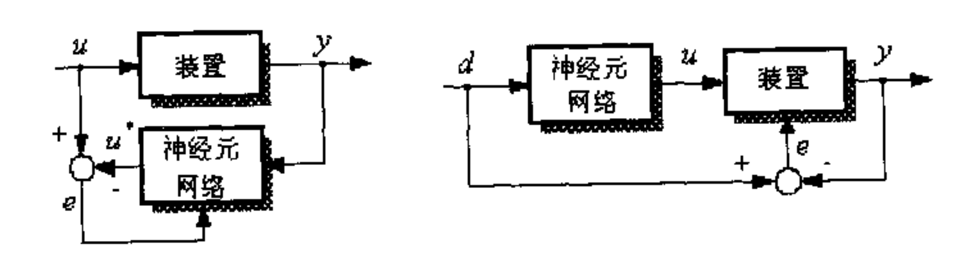


图 9-13　通用学习结构　　图 9-14　专用学习结构

3. 专用学习结构

多层 NN 控制的专用学习结构如图 9-14 所示。当神经网络训练时，期望输出 d 是该网络的输入。采用误差反向传播方法，经过训练使期望输出 d 与装置的实际输出 y 之间的差别 e 为最小。因此，不仅能够期望得到良好的装置输出，而且训练能够在期望输出范围内

执行，而不需要知道装置的合适输入范围。不过，本结构中把装置当做网络的一层来处理。为了训练该网络，或者必须知道装置的动力学模型，或者必须进行某种近似处理。对多层神经网络控制器的训练是由误差反向传播训练算法来完成的。该误差可为期望输出与实际装置输出间的差，也可为校正装置输入与由神经网络计算得到的输入之间的差。

9.9 分级 NN 控制

基于神经网络的分级控制模型如图 9-15 所示。

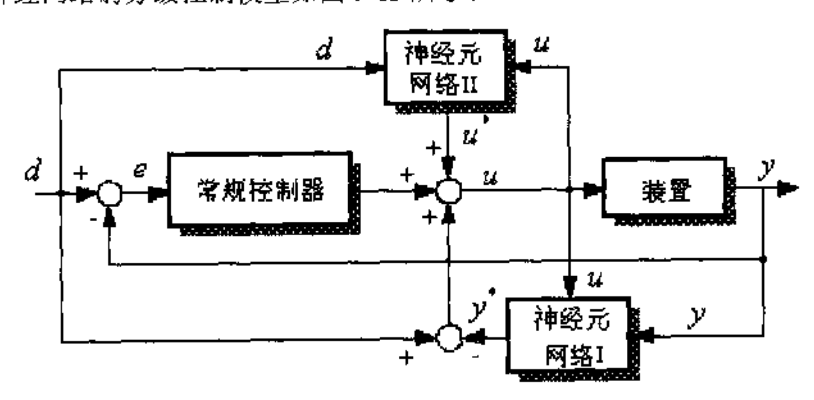


图 9-15　分级神经网络控制器

在图 9-15 中，d 为受控装置的期望输出，u 为装置的控制输入，y 为装置的实际输出，u^*和 y^*为由神经网络给出的装置计算输入与输出。该系统可视为由三部分组成。第一部分为一个常规外反馈回路。反馈控制是以期望装置输出 d 与由传感器测量的实际装置输出 y 间的误差 e 为基础的，即以 e=(d-y)为基础的。通常，常规外反馈控制器为一个比例微分控制器。

第二部分是与神经网络I连接的通道，该网络为一个受控对象的动力学内模型，用于监控装置的输入 u 和输出 y，且学习受控对象的动力学特性。当接收到装置的输入 u 时，经过训练，神经网络I就能够提供一个近似的装置输出 y^*。从这个意义上看，这部分起到系统动态特性辨识器的作用。以误差 $d-y^*$为基础，这部分提供一个比外反馈回路快得多的内反馈回路，因为外反馈回路一般在反馈通道上有传感滞后作用。

系统的第三部分是神经网络 II，它监控期望输出 d 和装置输入 u。这个神经网络学习建立装置的内动力学模型；当它收到期望输出指令 d 时，经过训练，它能够产生一个合适的装置输入分量 u^*。该受控对象的分级神经网络模型按下列过程运作。传感反馈主要在学习阶段起作用，此回路提供一个常规反馈信号去控制装置。由于传感延时作用和较小的可允许控制增益，因而系统的响应较慢，从而限制了学习阶段的速度。在学习阶段，神经网络 I 学习系统动力学特性，而神经网络 II 学习逆动力学特性。随着学习的进行，内反馈逐渐接替外反馈的作用，成为主控制器。然后，当学习进一步进行时，该逆动力学部分将取代内反馈控制。最后结果是，该装置主要由前馈控制器进行控制，因为装置的输出误差与内反馈一起几乎不复存在，从而提供处理随机扰动的快速控制。在上述过程中，控制与学

习同步执行。两个神经网络起到辨识器的作用，其中一个用于辨识装置动力学特性，另一个用于辨识逆动力学特性。

很重要的一点是，可把分级神经网络模型控制系统分为两个系统，即基于正向动力学辨识器的系统（见图 9-16）和基于逆向动力学辨识器的系统（见图 9-17），可以单独应用它们。

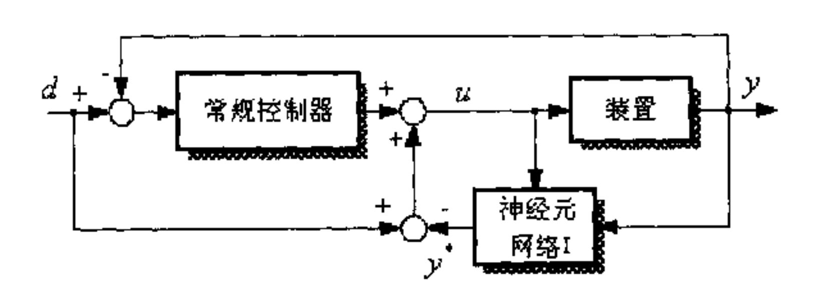


图 9-16　基于正向动力学辨识器的控制系统

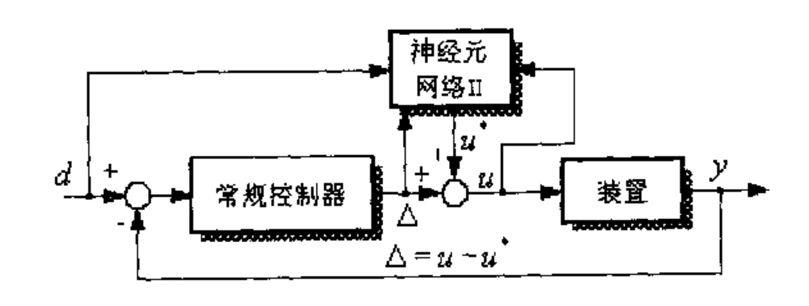
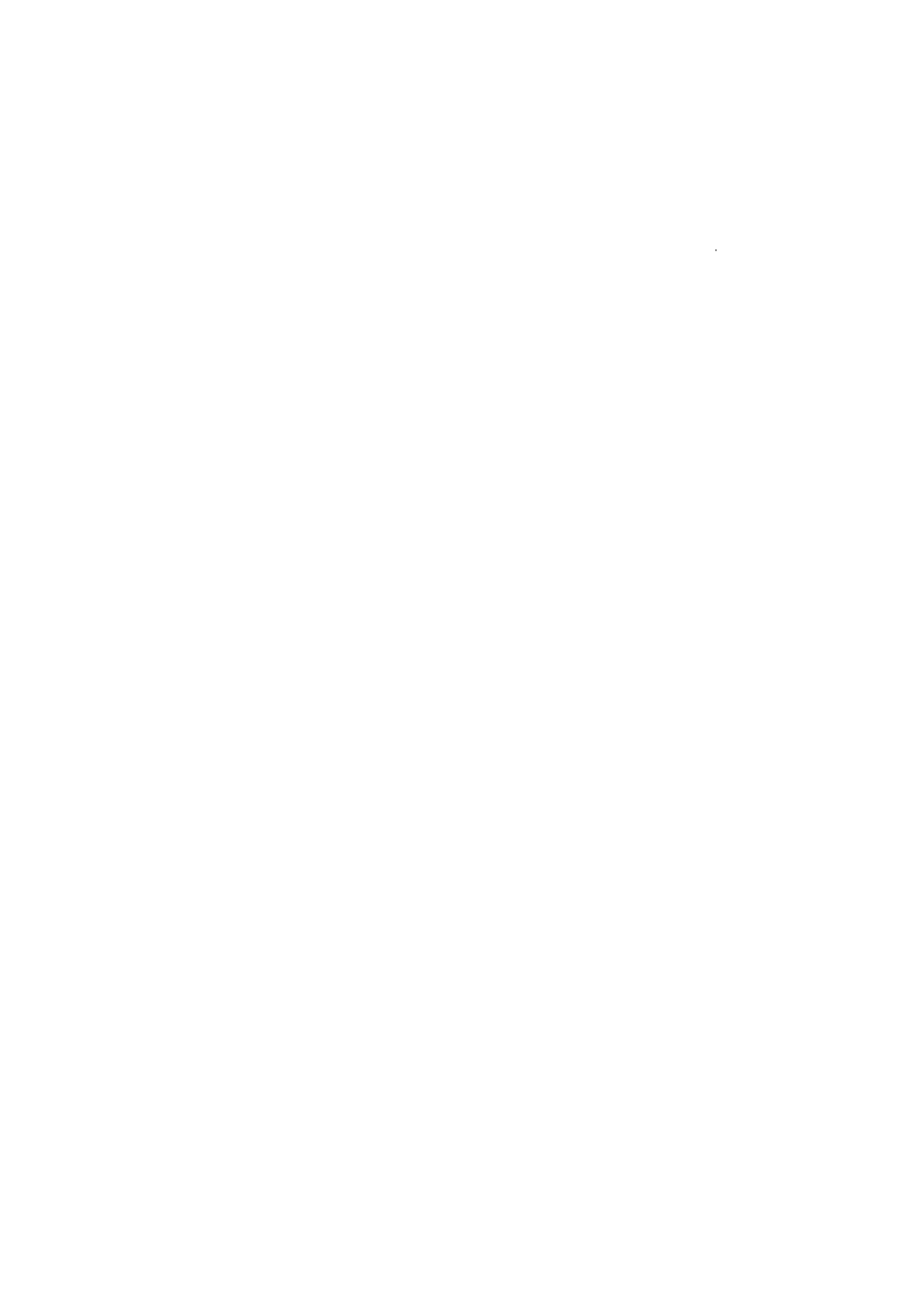


图 9-17　基于逆向动力学辨识器的控制系统

总之，基于分级神经网络模型的控制系统具有下列特点：

- 该系统含有两个辨识器，一个用于辨识装置的动力学特性，另一个用于辨识装置的逆动力学特性。
- 存在一个主反馈回路，它对训练神经网络是很重要的。
- 当训练进行时，逆动力学部分变为主控制器。
- 本控制的最后效果与前馈控制的效果相似。



第 10 章　神经网络控制理论及应用设计

神经网络在系统辨识和动态系统控制中已经得到了非常成功的使用。由于多层感知器网络具有全局逼近能力，使得其在对非线性系统建模和对一般情况下的非线性控制器的实现等方面应用得比较普遍。本章将介绍三种比较普遍的神经网络结构，它们常用于预测和控制，并且在 MATLAB 6.5 对应的神经网络工具箱中给出了实现。这三种神经网络结构分别是：

- 模型预测控制
- NARMA-L2（反馈线性化）控制
- 模型参考控制

使用神经网络进行控制时，通常有两个步骤：系统辨识和控制设计。

在系统辨识阶段，主要任务是对需要控制的系统建立神经网络模型；在控制设计阶段，主要使用神经网络模型来设计（训练）控制器。在本章将要介绍的三种控制网络结构中，系统辨识阶段是相同的，而控制设计阶段则各不相同。

对于模型预测控制，系统模型用于预测系统未来的行为，并且找到最优的算法，用于选择控制输入，以优化未来的性能。

对于 NARMA-L2（反馈线性化）控制，控制器仅仅是将系统模型进行重整。

对于模型参考控制，控制器是一个神经网络，它被训练以用于控制系统，使得系统跟踪一个参考模型。这个神经网络系统模型在控制器训练中起辅助作用。

本章主要内容：

- 模型预测控制理论
- 模型预测控制实例分析
- 反馈线性化控制理论
- NARMA-L2（反馈线性化）控制实例分析
- 模型参考控制
- 模型参考控制实例分析
- 总结

10.1　模型预测控制理论

神经网络预测控制器使用非线性神经网络模型来预测未来模型性能。控制器计算控制输入，而控制输入在未来一段指定的时间内将最优化模型性能。模型预测第一步是要建立

神经网络模型（系统辨识）；第二步，使用控制器来预测未来神经网络性能。

10.1.1 系统辨识

模型预测的第一步就是训练神经网络来表示网络的动态机制。模型输出与神经网络输出之间的预测误差，用来作为神经网络的训练信号，该过程用图 10-1 来表示。

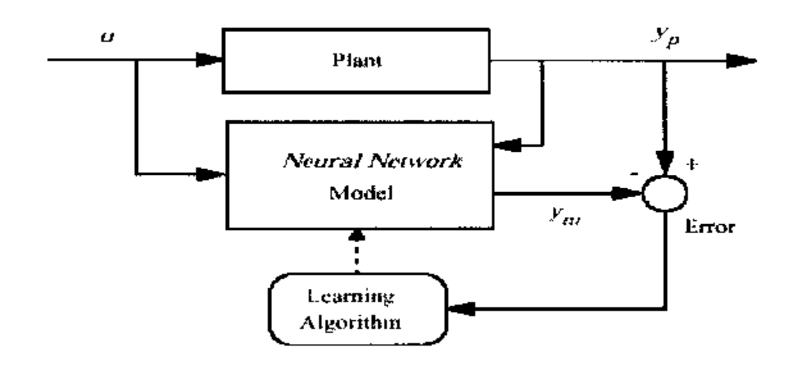


图 10-1　训练神经网络

神经网络模型利用当前输入和当前模型输出来预测网络未来输出值。神经网络模型结构如图 10-2 所示，该网络可以以批量方式进行在线训练。

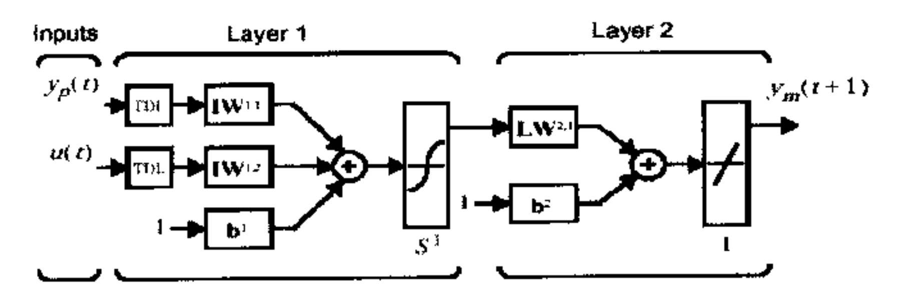


图 10-2　神经网络模型结构

10.1.2 模型预测

模型预测方法是基于水平后退方法的[SoHa96]，神经网络模型预测在指定时间内预测模型响应。预测使用数字最优化程序来确定控制信号，通过最优化如下的性能准则函数：

$$J=\sum_{N_1}^{N_2}(y_r(t+j)-y_m(t+j))^2+\rho\sum_{j=1}^{N_u}(u(t+j-1)-u(t+j-2))^2$$

在上式中，$N_1,N_2,\dots,N_u$ 定义范围，在该范围内计算跟踪误差和控制增益。变量 u 是实验控制信号，y_r 是期望响应，y_m 是网络模型响应，ρ 值反映了控制增益的平方和的

分布。

图 10-3 描述了模型预测控制的过程。控制器由神经网络模型和最优化方块组成，最优化方块可确定 u（通过最小化 J），最优 u 值作为神经网络模型的输入，控制器方块可在 simulink 实现。

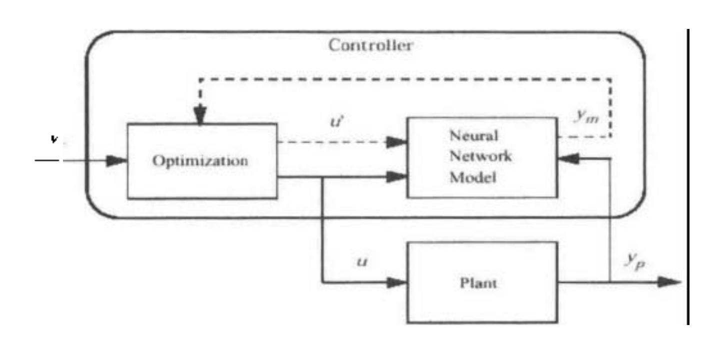


图 10-3　模型预测控制的过程

10.2　模型预测控制实例分析

在 MATLAB 6.5 神经网络工具箱中实现的神经网络预测控制器使用了一个非线性系统模型，用于预测系统未来的性能。接下来这个控制器将计算控制输入，用于在某个未来的时间区间里优化系统的性能。进行模型预测控制首先要建立系统的模型，然后使用控制器来预测未来的性能。下面将结合 MATLAB 6.5 经网络工具箱中提供的一个演示实例，介绍其在 SIMULINK 中的实现过程。

10.2.1　问题的描述

要讨论的问题基于一个搅拌器（CSTR），如图 10-4 所示。

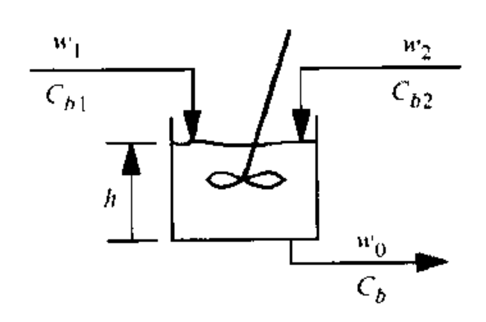


图 10-4　搅拌器

对于这个系统，其动力学模型为：

$$\frac{\mathrm{d}h(t)}{\mathrm{d}t} = w_1(t) + w_2(t) - 0.2\sqrt{h(t)}$$

$$\frac{\mathrm{d}C_b(t)}{\mathrm{d}t} = (C_{b1} - C_b(t))\frac{w_1(t)}{h(t)} + (C_{b2} - C_b(t)\frac{w_2(t)}{h(t)} - \frac{k_1 C_b(t)}{(1 + k_2 C_b(t))^2}$$

其中 $h(t)$ 为液面高度，$C_b(t)$ 为产品输出浓度，$w_1(t)$ 为浓缩液 C_{b1} 的输入流速，$w_2(t)$ 为稀释液 C_{b2} 的输入流速。输入浓度设定为：$C_{b1} = 24.9$，$C_{b2} = 0.1$。消耗常量设置为：$k_1 = 1$，$k_2 = 1$。

控制的目标是通过调节流速 $w_2(t)$ 来保持产品浓度。为了简化演示过程，不妨设 $w_1(t) = 0.1$。在本例中不考虑控制液面高度 $h(t)$。

10.2.2 建立模型

在 MATLAB 6.5 神经网络工具箱 NN Toolbox 4.0.2 中提供了这个演示实例。只需在 MATLAB 命令行中输入 predcstr，就会自动地调用 SIMULINK，并且产生如图 10-5 所示的模型窗口。神经网络预测控制器模块已经被放置在这个模型中。

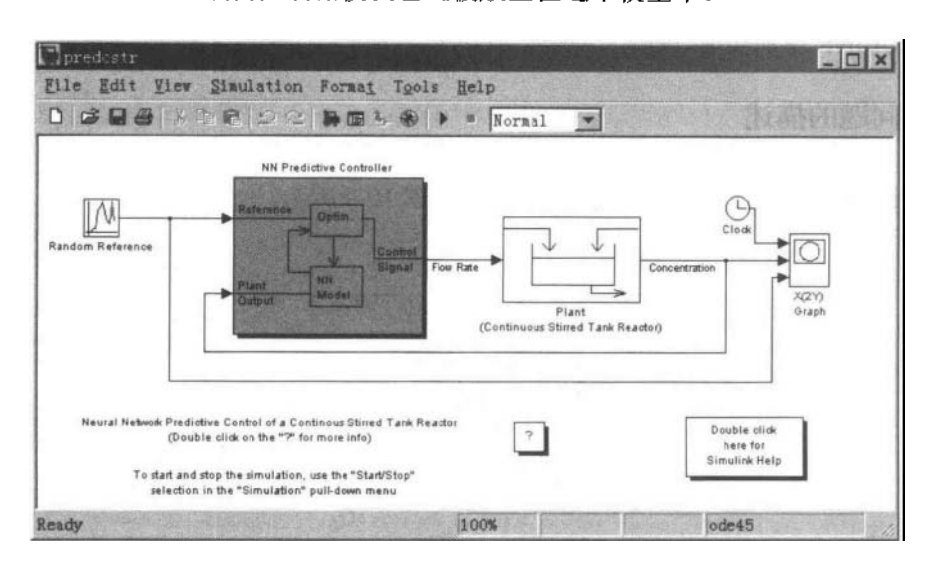


图 10-5 模型窗口

图 10-5 中的 Plant（Continuous Stirred Tank Reactor）模块包含了系统模型的 SIMULINK 模型。双击这个模块，可以得到其具体的 SIMULINK 实现，如图 10-6 所示。

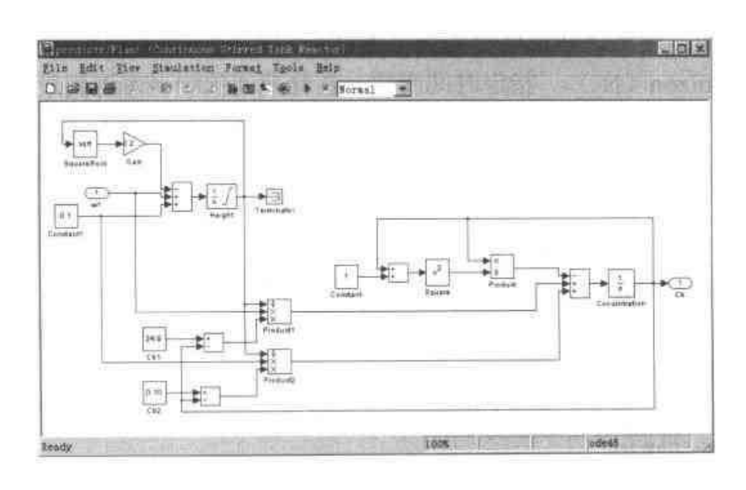


图 10-6　搅拌器系统的 SIMULINK 模型

关于这个控制系统的模型描述不是本书的重点，此处将不加以深入讨论，有兴趣的读者可以参考 MATAB 控制系统设计的相关书籍。

图 10-5 中有一个 NN Predictive Controller 模块，这个模块是在神经网络工具箱中生成并复制过来的。这个模块的 Control Signal 端连接到系统模型的输入端，系统模型的输出端连接到模块的 Plant Output 端，参考信号连接到模块的 Reference 端。

双击 NN Predictive Controller 模块，将会产生一个新的窗口，如图 10-7 所示。这个窗口用于设计模型预测控制器，调整相关的参数。在这个窗口中，我们可以改变 N_2 和 N_u 的值。另外，权重参数 ρ 也可以在这里调整。参数 α 也能被调整，它用于控制最优性，它决定了对于一个成功的优化步骤，在性能上需要多大的阻尼。在这个窗口中，还可以选择使用哪种线性最小化程序，用于优化算法。另外，还可以设定在每个采样时间里进行多少次迭代优化算法。

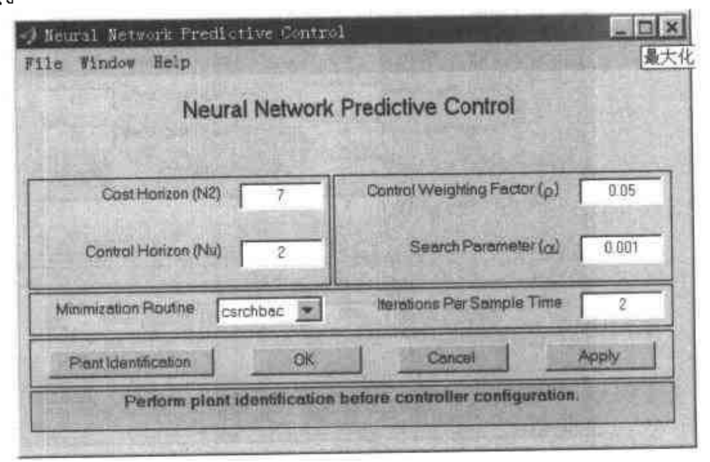


图 10-7　模型预测控制器参数设置窗口

对于图 10-7 所示的窗口，有多项参数可以调整。将鼠标移到相应的位置，就会出现

对这一参数的说明。现将这些说明分别加以解释。

- Cost Honizon（N_2） 指定时间步数，在此期间预测误差达到最小。
- Control Honizon（N_u） 指定时间步数，在此期间控制增量达到最小。
- Minimization Routine 可以从几个线性搜索程序中选择一个用做最优化算法。
- Control Weight Factor（ρ） 在性能函数中，控制权重因子用于与控制增量的平方和相乘。
- Search Parameter（α） 这个参数决定了线性搜索何时停止。
- Iterations Per Sample Time 选择在每个采样时间中优化算法迭代的次数。

下面是四个按钮的说明：

- Plant Identification 单击此按钮可打开系统辨识窗口。在控制器使用之前，系统必须先进行辨识。
- OK、Apply 在控制器参数设定好以后，单击这两个按钮的任一个都可以将这些参数导入 SIMULINK 模型。
- Cancel 取消刚才的设置。

10.2.3 系统辨识

单击【Plant Identification】按钮，将产生一个新的窗口，用于设置系统辨识的参数，如图 10-8 所示。

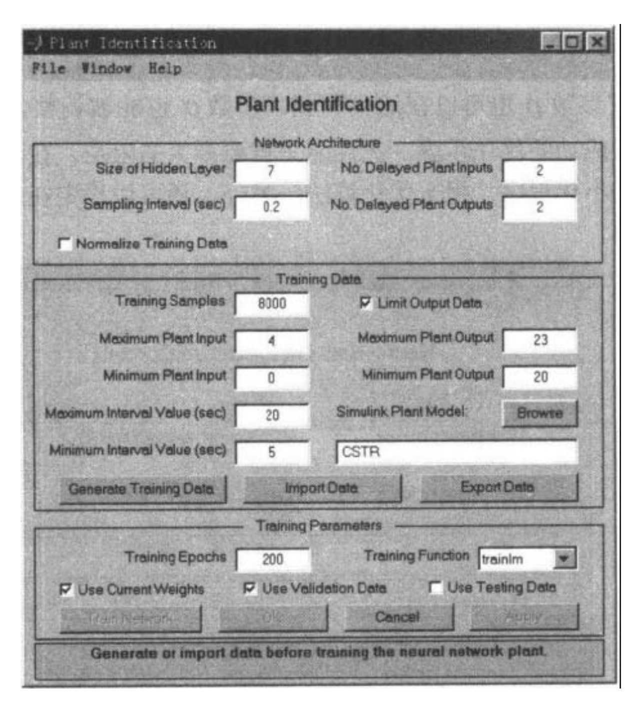


图 10-8 系统辨识参数设置窗口

在控制器使用以前，必须建立神经网络模型。这个模型预测系统未来的输出值。优化

算法使用这些预测值来决定控制输入，以优化未来的性能。系统的神经网络模型有一个隐含层。这个隐含层的大小、输入和输出的时延，以及训练函数都在图 10-8 所示的窗口中设置。可以选择 BP 网络中的任意训练函数来训练网络模型。

与图 10-7 类似，在图 10-8 所示的窗口中，有很多参数需要设置。将鼠标移到相应的位置，也会出现对这些参数的说明。现将这些参数分别加以解释。

在窗口菜单中有一项 File，其包含的子项中有两项用于导入和导出系统模型对应的网络。

- Size of Hidden Layer　设置在系统模型网络第一层中的神经元数目。
- Sampling Interval（sec）　指定程序从 SIMULINK 模型中采集数据的间隔。
- Normalize Training Data　指定是否使用 premnmx 函数来将数据标准化。
- No.Delayed Plant Inputs　指定了加到系统网络模型的输入延迟。
- No.Delayed Plant Outputs　指定了加到系统网络模型的输出延迟。
- Training Samples　指定了为训练而产生的数据点的数目。
- Maximum Plant Input　指定了随机输入的最大值。
- Minimum Plant Input　指定了随机输入的最小值。
- Maximum Interval Value（sec）　指定一个最大的间隔，在这个间隔中，随机输入将保持不变。
- Minimum Interval Value（sec）　指定一个最小的间隔，在这个间隔中，随机输入将保持不变。
- Limit Output Data　用于选择系统输出是否为有界值。
- Maximum Plant Output　指定了输出的最大值。
- Minimum Plant Output　指定了输出的最小值。
- Simulink Plant Model　指定用于产生训练数据的模型（.mdl 文件）。
- Training Epochs　指定训练迭代的次数。
- Training Function　指定训练函数。
- Use Current Weights　指定是否选择当前的权重用于连续训练。
- Use Valid Data　指定是否选择合法数据停止训练。
- Use Testing Data　指定在训练过程中测试数据是否被追踪。

下面是关于按钮的说明：

- Generate Training Data　产生用于网络训练的数据。
- Import Data　从工作空间或者一个文件中导入数据。
- Export Data　将训练数据导出到工作空间或者一个文件中。
- Train Network　开始网络模型的训练。在训练前必须已经产生或者导入了数据。
- OK、Apply　在网络模型经过训练后，单击这两个按钮中的任一个都可以将网络导入 SIMULINK 模型。
- Cancel　取消刚才的设置。

单击【Generate Training Data】按钮，程序就会通过对 SIMULINK 网络模型提供一系列随机阶跃信号，来产生训练数据，图 10-9 显示了这些训练数据。

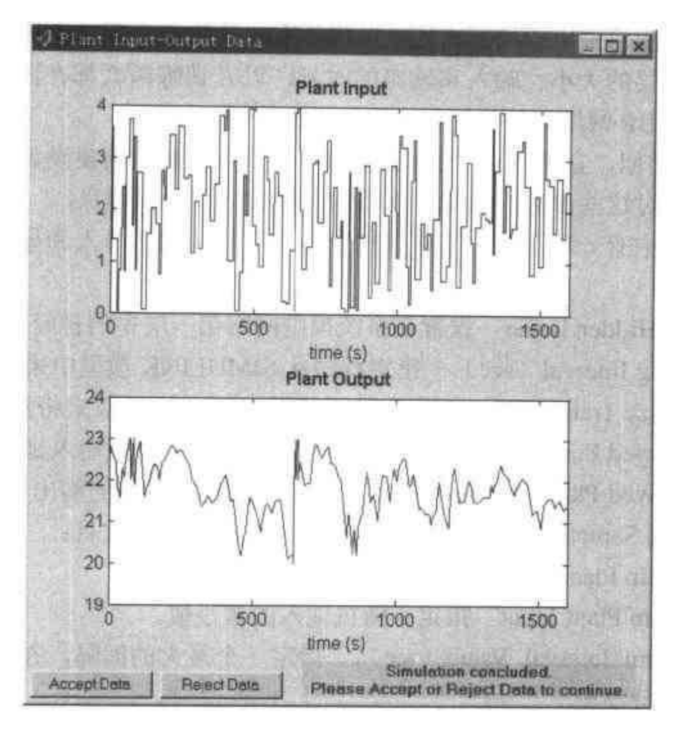


图 10-9　训练数据

在图 10-9 中，有两个按钮，一个为【Accept data】按钮，如果单击这个按钮，那么就接受了这些训练数据，另一个为【Refuse Data】按钮，如果单击这个按钮，将会回到系统辨识窗口，并且可以重新开始训练。

单击【Accept Data】按钮，然后在系统辨识窗口中单击【Train Network】按钮。网络模型开始训练。训练与选择的训练算法有关（在本处使用的是 trainlm）。在训练结束后，相应的结果被显示出来，如图 10-10（a）及 10-10（b）所示。

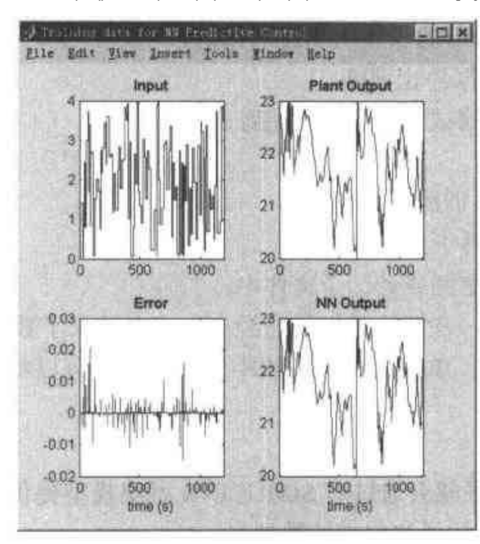


图 10-10（a）训练数据

图 10-10（b）合法数据

图 10-10（a）显示的是训练数据，图 10-10（b）显示的是合法数据。在每个图中，左上角的图显示了随机输入：随机的阶跃高度和宽度；右上角的图显示了模型 SIMULINK 输出；左下角的图显示了误差，即系统输出与网络模型输出的差别；右下角的图显示了神经网络模型输出。

此时，可以单击【Train Network】按钮继续再次使用同样的数据进行训练，也可以单击【Erase Generated Data】按钮，并且产生新的数据。当然，也可以接受当前的模型，并且开始对闭环系统进行仿真。

10.2.4 系统仿真

在系统辨识窗口中单击【OK】按钮，将训练好的神经网络模型导入到 NN Predictive Controller 模块中。

在神经网络预测控制窗口中单击【OK】按钮，将控制器参数导入到 NN Predictive Controller 模块中。

返回到 SIMULINK 模型，并且从【Simulation】菜单中单击【Start】命令开始仿真。仿真的过程需要一段时间。当仿真结束时，将会显示出系统的输出，同时，参考信号也会显示出来，如图 10-11 所示。

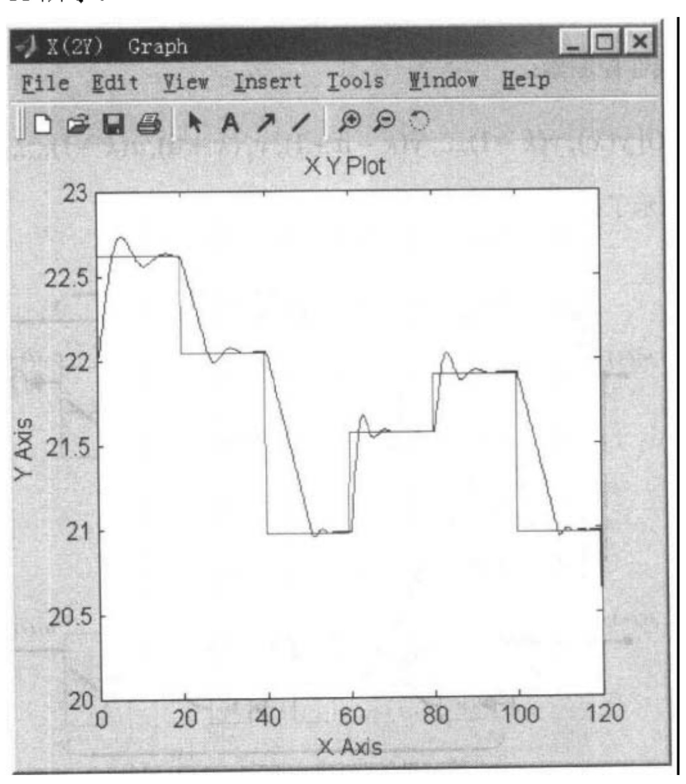


图 10-11 仿真得到的系统输出及参考信号

10.3 反馈线性化控制理论

这一节描述的神经网络控制器有两个不同的名字：反馈线性化控制器和 NARMA-L2 控制器。这类控制器的中心思想是：通过去掉非线性，将一个非线性系统变换成线性系统。

10.3.1 辨识 NARMA-L2 模型

与模型预测控制一样，反馈线性化控制的第一步就是去辨识被控制的系统。通过训练一个神经网络来表示系统的前向动态机制，第一步首先选择一个模型结构以供使用。一个用来代表一般的离散非线性系统的标准模型是：非线性自回归移动平均模型（NARMA），用下式表示：

$$y(k+\mathrm{d}) = N[y(k), y(k-1), ..., y(k-n+1), u(k), u(k-1), ..., u(k-n+1)]$$

式中，$u(k)$ 表示系统的输入，$y(k)$ 表示系统的输出。在辨识阶段，训练神经网络使其近似等于非线性函数 N。

如果希望系统输出跟随一些参考曲线 $y(k+\mathrm{d}) = y_r(k+\mathrm{d})$，下一步就是建立一个有如下形式的非线性控制器：

$$u(k) = G[y(k), y(k-1), ..., y(k-n+1), y_r(k+\mathrm{d}), u(k-1), ..., u(k-n+1)]$$

图 10-12 表示了神经网络结构。

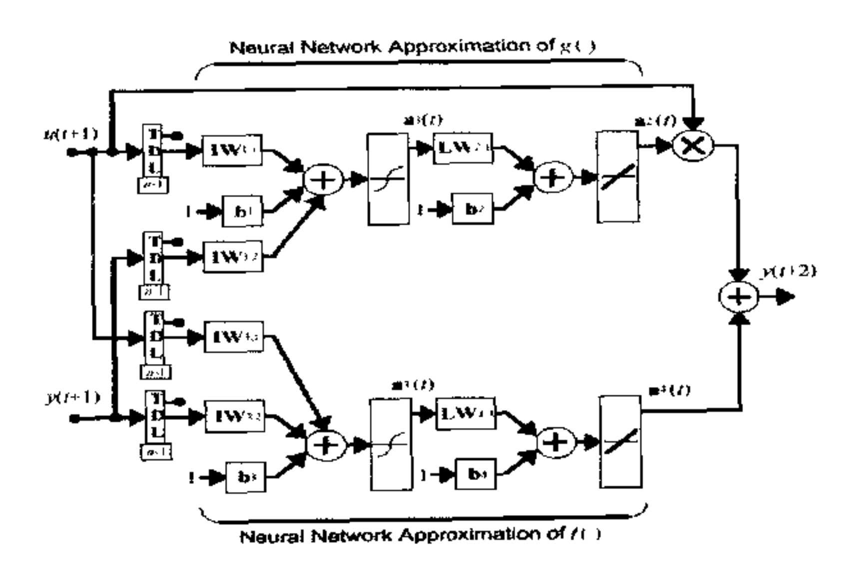
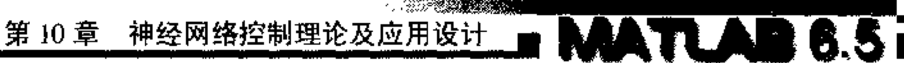


图 10-12 神经网络结构

使用该类控制器的问题是：如果您想训练一个神经网络用来产生函数 G（最小化均方



差），您必须使用动态反馈，且该过程相当慢。由 Narendra 和 Mukhopadhyay 提出的一个解决办法是：使用近似模型来代表系统。在这里使用的控制器模型是基于 NARMA-L2 近似模型：

$$\hat{y}(k+\mathrm{d}) = f[y(k), y(k-1),\ldots, y(k-n+1), u(k-1),\ldots,u(k-n+1)] + g[y(k), y(k-1),\ldots, y(k-n+1), u(k-1),\ldots, u(k-n+1)]$$

该模型是并联形式，下一个控制器输入 $u(k)$ 没有包含在非线性系统里。这种形式的优点是：能解决控制器输入使系统输出跟随参考曲线 $y(k+\mathrm{d}) = y_\tau(k+\mathrm{d})$。最终的控制器形式如下：

$$u(k) = \frac{y_\tau(k+d) - f[y(k), y(k-1),\ldots, y(k-n+1), u(k-1),\ldots, u(k-n+1)]}{g[y(k), y(k-1),\ldots, y(k-n+1), u(k-1),\ldots, u(k-n+1)]}$$

直接使用该等式会引起实现问题，因为基于输出 $y(k)$ 的同时您必须同时得到 $u(k)$，所以我们采用下述模型：

$$y(k+\mathrm{d}) = f[y(k),\ldots, y(k-n+1), u(k),\ldots,u(k-n+1)] + g[y(k),\ldots, y(k-n+1), u(k),\ldots, u(k-n+1)]\cdot u(k-1)$$

这里d≥2。

10.3.2　NARMA-L2 控制器

利用 NARMA-L2 模型，可得到如下的控制器：

$$u(k+1) = \frac{y_\tau(k+\mathrm{d}) - f[y(k), y(k-1),\ldots, y(k-n+1), u(k-1),\ldots, u(k-n+1)]}{g[y(k), y(k-1),\ldots, y(k-n+1), u(k-1),\ldots, u(k-n+1)]}$$

这里d ≥ 2， 图 10-13 表示 NARMA-L2 模型控制器。

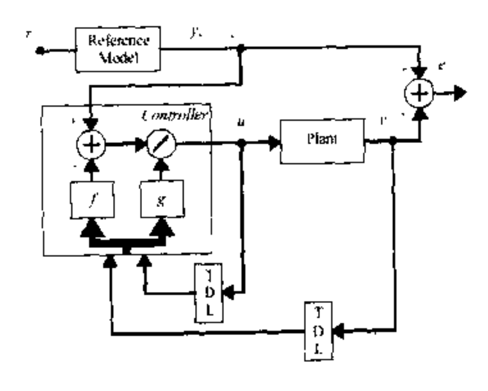


图 10-13　NARMA-L2 模型控制器

该控制器可在已辨识出的 NARMA 实验模型里得到实现，如图 10-14 所示。

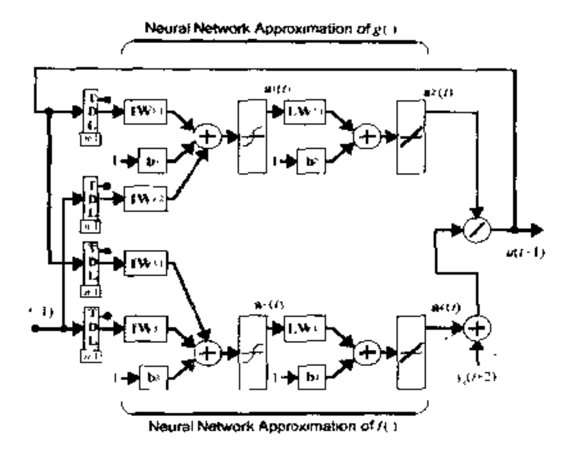


图 10-14　控制器实现

10.4　NARMA-L2（反馈线性化）控制实例分析

NARMA-L2（反馈线性化）控制的主要思想是通过去掉非线性部分，将非线性动态系统转换成线性动态系统。本节首先将先给出一个相伴型系统模型，并介绍怎样使用神经网络来辨识这个模型，然后描述怎样使用辨识了的神经网络模型来设计控制器。下面将结合 MATLAB 6.5 神经网络工具箱中提供的一个实例，来介绍这一过程。

10.4.1　问题的描述

如图 10-15 所示，有一块磁铁，被约束在垂直方向上运动。在其下方有一块电磁铁，通电以后，点磁铁就会对其上的磁铁产生电磁力作用。我们的目标就是通过控制电磁铁，使得其上的磁铁保持悬浮在空中，不会掉下来。

建立这个实际问题的动力学方程式如下：

$$\frac{\mathrm{d}^2 y(t)}{\mathrm{d}t^2} = -g + \frac{\alpha i^2(t)}{My(t)} - \frac{\beta}{M}\frac{\mathrm{d}\,y(t)}{\mathrm{d}t}$$

图 10-15　悬浮磁铁控制

上式中 $y(t)$ 表示磁铁离电磁铁的距离，$i(t)$ 代表电磁铁中的电流，M 代表磁铁的质量，g 代表重力加速度。参数 β 代表粘性摩擦系数，它由磁铁所在的容器的材料决定；α 代表场强常数，它由电磁铁上所绕的线圈圈数，以及磁铁的强度所决定。

10.4.2 建立模型

MATLAB 6.5 的神经网络工具箱 NN Toolbox 4.0.2 提供了这个演示实例。只需在 MATLAB 命令行中输入 narmamaglev，就会自动地调用 SIMULINK，并且产生如图 10-16 所示的模型窗口。NARMA-L2 控制模块已经被放置在这个模型中。

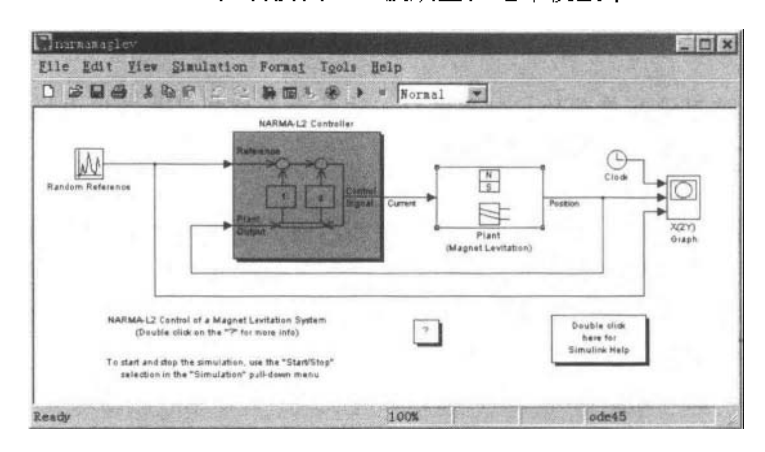


图 10-16 模型窗口

与模型预测控制一样，图 10-16 中的 Plant（Magnet Levitation）模块包含了磁悬浮系统的 SIMULINK 模型。同样，只需双击这个模块，便可以得到其具体的 SIMULINK 实现，如图 10-17 所示。

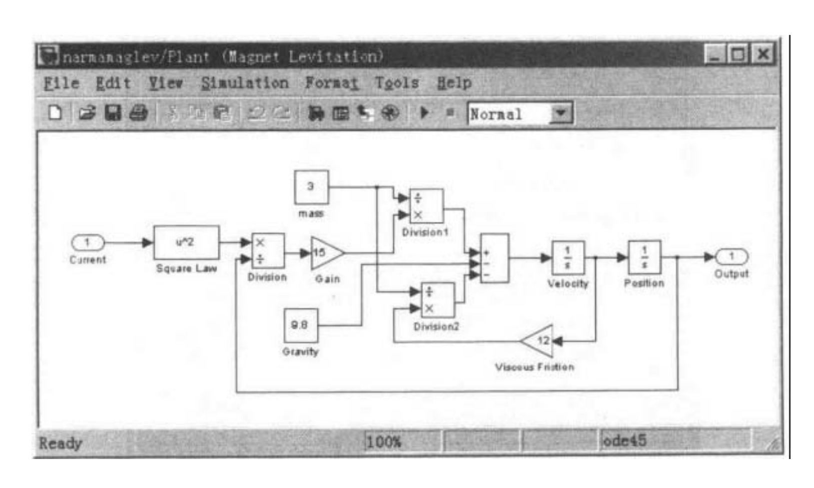


图 10-17 磁悬浮系统的 SIMULINK 模型

关于这个磁悬浮系统的 SIMULINK 模型的建立过程并不是本书的重点，故此处只是加以引用，而不打算进行更深入的探讨，有兴趣的读者可以参考 MATAB 控制系统 SIMULINK 设计的相关书籍。

下面回到图 10-16 所示的窗口中。在这个窗口中有一个 NARMA-L2 Controller 模块，这个模块是在神经网络工具箱中生成并复制过来的。这个模块的 Control Signal 端连接到此悬浮系统模型的 Current 输入端，系统模型的 Position 输出端连接到模块的 Plant Output 端，参考信号连接到模块的 Reference 端。

10.4.3 系统辨识

双击 NARMA-L2 Controller 模块，将会产生一个新的窗口，如图 10-18 所示。这个窗口用于训练 NARMA-L2 模型。



这里没有单独的控制器窗口，这是由于控制器是直接由模型得到的，在这一点上与模型预测控制不同。

与模型预测控制类似，在使用神经网络控制之前，必须先对系统进行辨识。图 10-18 所示的窗口，即为系统辨识参数设置窗口，其中有很多参数需要设置。与前面介绍过的一样，将鼠标移到相应的位置，也会出现对这些参数的说明。由于前面已经给出了较详细的介绍，本处就不再赘述，读者可以参照上一节的相关部分。

在进行仿真之前，先要训练网络，而训练前需要产生相应的训练数据。

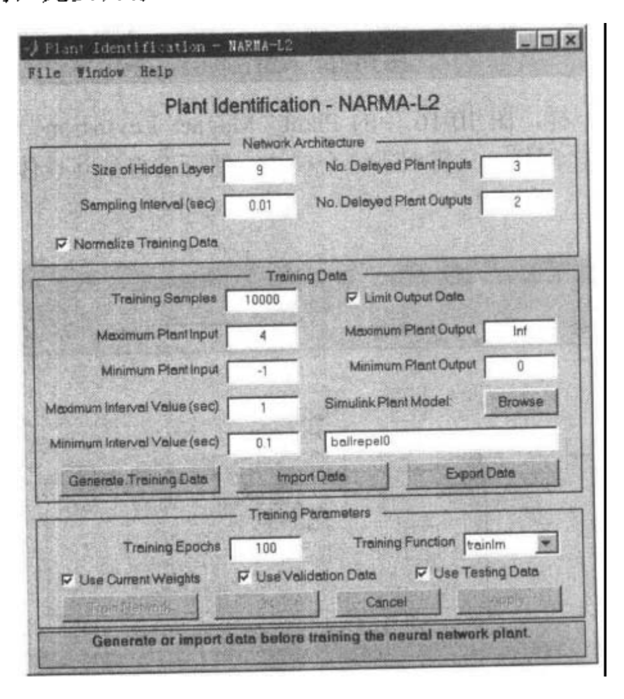


图 10-18　系统辨识参数设置窗口

同 10.2 节类似，单击【Generate Training Data】按钮，产生训练数据。相应的过程可参考 10.2 节的对应部分，本处只给出结果，如图 10-19 所示。

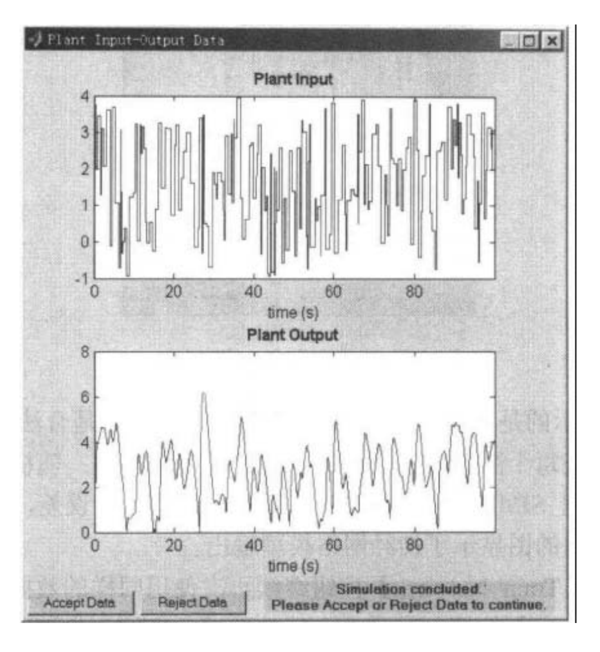
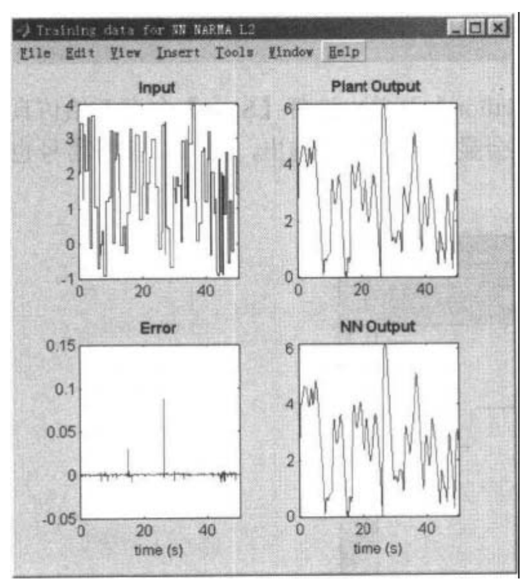
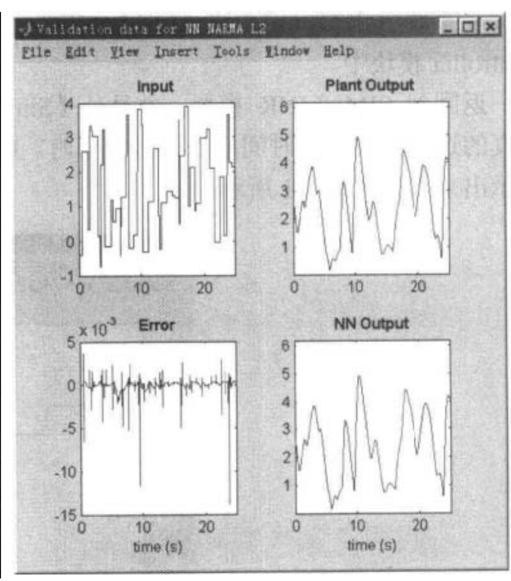


图 10-19　训练数据

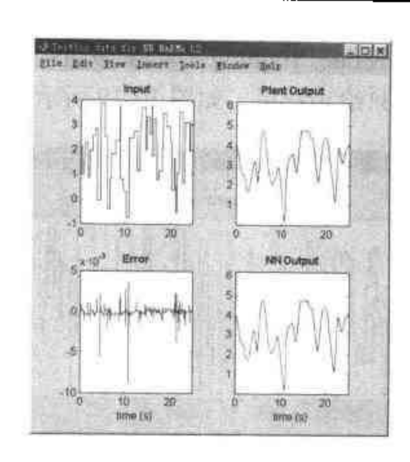
同前所述，接受这些数据，并开始训练。此处给出相应的训练结果分别如图 10-20（a）～图 10-20（c）所示。







10-20（a）　训练情况　　　　10-20（b）　合法数据训练情况



10-20（c） 测试数据

图 10-20（a）显示的是训练数据，图 10-20（b）显示的是合法数据，图 10-20（c）显示的是测试数据。在每个图中，左上角的图显示了随机输入：随机的阶跃高度和宽度；右上角的图显示了模型 SIMULINK 输出；左下角的图显示了误差，即系统输出与网络模型输出的差别；右下角的图显示了神经网络模型输出。

此时，可以单击【Train Network】按钮继续再次使用同样的数据进行训练，也可以单击【Erase Generated Data】按钮，并且产生行的数据。当然，也可以接受当前的模型，并且开始对闭环系统进行仿真。

10.4.4 系统仿真

在系统辨识窗口中单击【OK】按钮，将训练好的神经网络模型导入到 NARMA-L2 Controller 模块中。

返回到 SIMULINK 模型，并且从【Simulation】菜单中选择【Start】命令开始仿真。仿真的过程需要一段时间。当仿真结束时，将会显示出系统的输出，同时，参考信号也会显示出来，如图 10-21 所示。

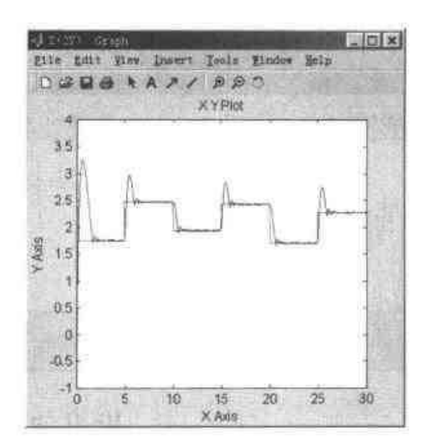


图 10-21 仿真得到的系统输出及参考信号



10.5 模型参考控制理论

神经模型参考控制采用两个神经网络：一个控制器网络和一个实验模型网络，如图 10-22 所示。首先辨识出实验模型，然后训练控制器，使得实验输出跟随参考模型输出。

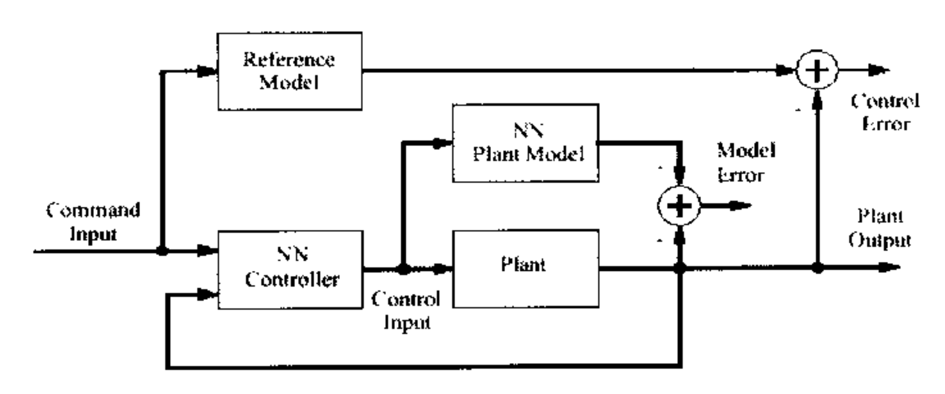


图 10-22 神经模型参考控制

图 10-23 显示了神经网络实验模型的详细情况，它们可以从神经网络工具箱中得到。每个网络有二层，在隐层里您可选取神经元的数目来使用。

这里存在三类控制器输入：

- 延迟参考输入
- 延迟控制输出
- 延迟实验输出

关于控制器，您可选择延迟数目。如何设置这些参数将在其他地方描述。

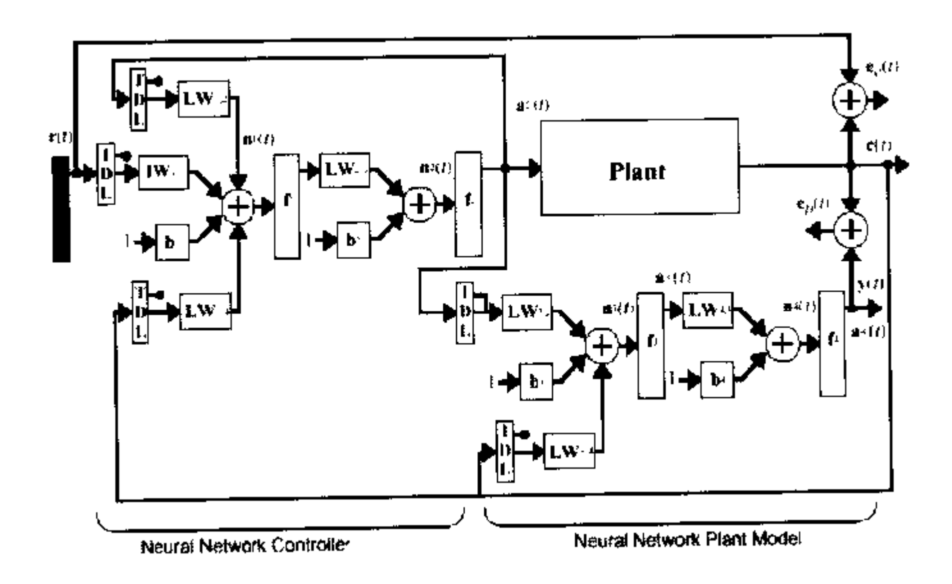


图 10-23 神经网络实验模型

10.6 模型参考控制实例分析

神经网络模型参考控制体系结构使用两个神经网络：一个控制器神经网络和一个系统模型神经网络。首先，对系统模型进行辨识，然后，训练控制器神经网络，使得系统输出跟踪参考模型的输出。

MATLAB 6.5 的神经网络工具箱 NN Toolbox 4.0.2 提供了这种神经网络的实现。每个网络由两层组成，并且可以选择隐含层的神经元数目。有三组控制器输入：

- 延迟的参考输入
- 延迟的控制输出
- 延迟的系统输出

对于每一个这种输入，可以选择延迟值。通常，随着系统阶次的增加，延迟的数目也增加。对于神经网络系统模型，有两组输入：

- 延迟的控制器输出
- 延迟的系统输出

下面结合 MATLAB 神经网络工具箱中提供的一个实例，来介绍神经网络控制器的训练过程。

10.6.1 问题的描述

图 10-24 中显示了一个简单的单连接机械臂，我们的目的是控制它的运动。

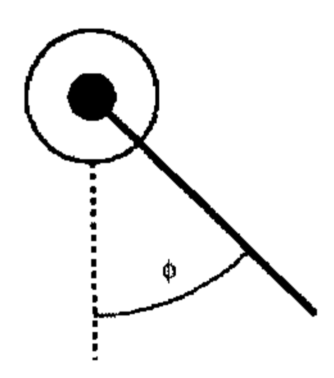


图 10-24 简单的单连接机械臂

首先，建立它的运动方程式，如下所示：

$$\frac{d^2\Phi}{dt^2} = -10\sin\Phi - 2\frac{d\Phi}{dt} + u$$

在这里，Φ 代表机械臂的角度，u 代表 DC（直流）电机的转矩。我们的目标是训练控制器，使得机械臂能够跟踪参考模型：

$$\frac{d^2 y_r}{dt^2} = -9y_r - 6\frac{dy_r}{dt} + 9r$$

这里 y_r 代表参考模型的输出，r 代表参考信号。

10.6.2　模型的建立

MATLAB 6.5 的神经网络工具箱 NN Toolbox 4.0.2 提供了这个演示实例。其中，使用的神经网络控制器为 5-13-1 结构。控制器的输入包含了两个延迟参考输入、两个延迟系统输出和一个延迟控制器输出，采样间隔为 0.05 秒。

只需在 MATLAB 命令行中输入 mrefrobotarm，就会自动地调用 SIMULINK，并且产生如图 10-25 所示的模型窗口。模型参考控制模块已经被放置在这个模型中。

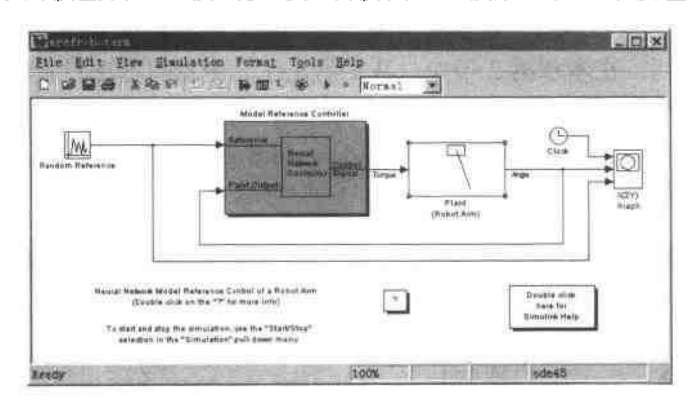


图 10-25　模型窗口

与前面的模型预测控制一样，图 10-25 中的 Plant（Robot Arm）模块包含了机械臂系统的 SIMULINK 模型。同样，只需双击这个模块，便可以得到其具体的 SIMULINK 实现，如图 10-26 所示。

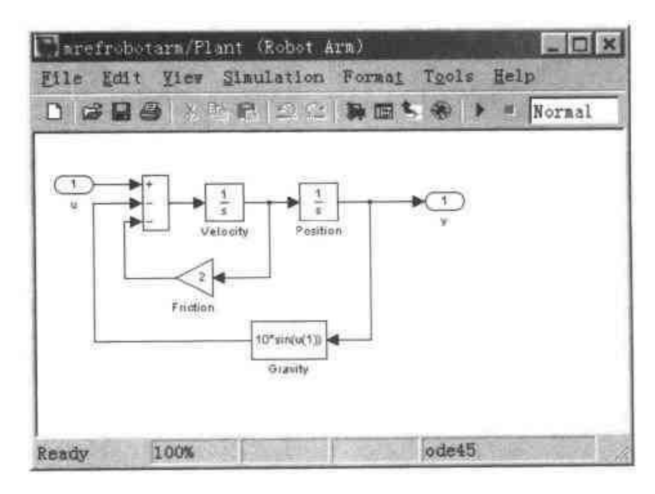


图 10-26　机械臂的 SIMULINK 模型

关于这个机械臂系统的 SIMULINK 模型的建立过程并不是本书的重点，故此处只是加以引用，而不打算进行更深入的探讨，有兴趣的读者可以参考 MATAB 控制系统 SIMULINK 设计的相关书籍。

下面回到图 10-25 所示的窗口中。在这个窗口中有一个 Model Reference Controller 模块，这个模块是在神经网络工具箱中生成并复制过来的。这个模块的 Control Signal 端连接到此机械臂系统模型的 Torque 输入端，系统模型的 Angle 输出端连接到模块的 Plant Output 端，参考信号连接到模块的 Reference 端。

10.6.3 系统辨识

双击模型参考控制模块，将会产生一个新的窗口，如图 10-27 所示。这个窗口用于训练模型参考控制器。

对于图 10.-27 所示的窗口，有多项参数可以调整。将鼠标移到相应的位置，就会出现对这一个参数的说明。在本章第 2 节中，我们给出了几个窗口参数设置的较详细的解释，此处与前面类似，故不再解释，读者可以参考前面的解释。

下一步在图 10-27 所示的窗口中单击【Plant Identification】按钮，系统辨识窗口将会弹出，如图 10-28 所示。

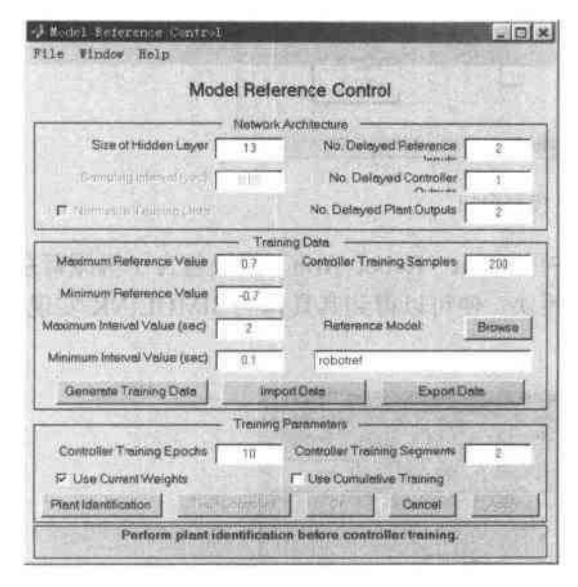


图 10-27　模型参考控制窗口

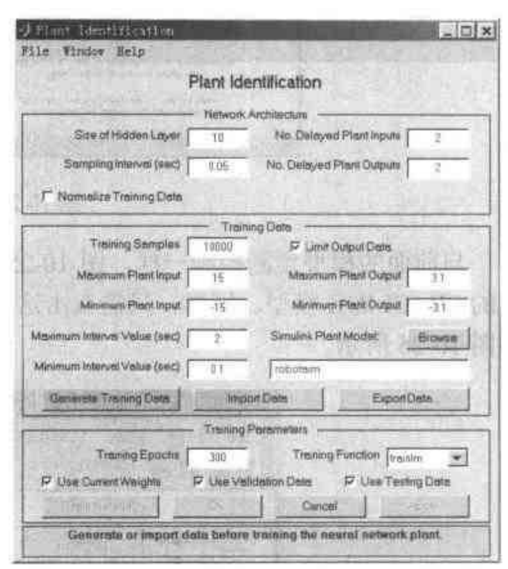


图 10-28　系统辨识窗口

由于在前面两节的设计中均有关于系统辨识窗口的操作，故本处不再赘述。

下一步单击【Generate Data】按钮，程序开始产生训练控制器需要的数据，在数据产生结束后，将会出现如图 10-29 所示的数据窗口。

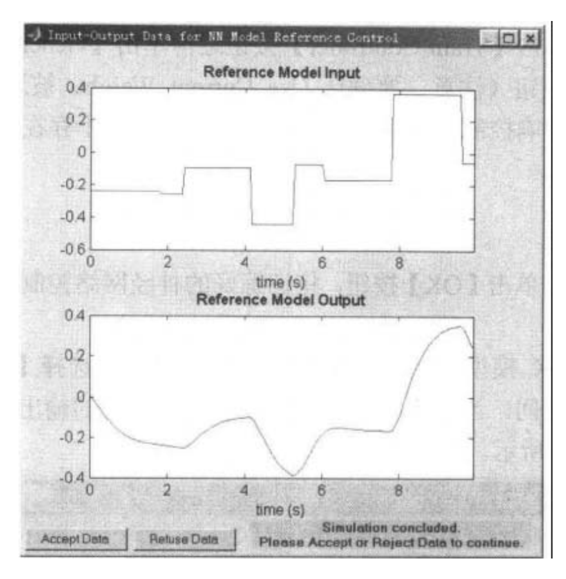


图 10-29 训练数据

单击【Accept Data】按钮，返回到模型参考控制窗口。单击【Train Controller】按钮，开始训练。程序将一段数据输入网络，并且对网络进行指定数目的迭代（在这里为 5）。直到所有的训练数据都输入了网络，这个过程才停止。控制器训练需要的时间比系统模型训练需要的时间长得多。这是因为控制器必须使用动态反传算法。训练结束后，闭环系统的响应结果被显示出来，如图 10-30 所示。

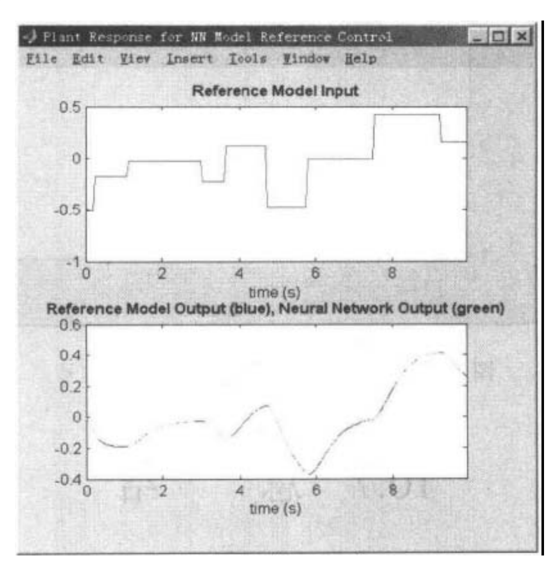


图 10-30 闭环响应结果

图 10-30 中上部分表示的是用于训练的随机参考信号，下部分表示的是参考模型的响应信号，以及闭环系统的响应信号。系统的响应信号应该跟踪参考模型的信号。

回到模型参考控制窗口，如果控制器的性能不准确，那么，可以再次单击【Train Controller】按钮，这样就会继续使用同样的数据对控制器进行训练。如果需要使用新的数

据继续训练，可以在单击【Train Controller】按钮之前单击【Generate Training Data】按钮或者【Import Data】按钮（注意，要确认 Use Current Weights 被选中）。另外，如果系统模型不够准确，也会影响控制器的训练。当然，在本例中，不存在这个问题。

10.6.4 系统仿真

在系统辨识窗口中单击【OK】按钮，将训练好的神经网络控制器权重导入 SIMULINK 模型中。

返回到 SIMULINK 模型，并且从【Simulation】菜单中选择【Start】命令开始仿真。仿真的过程需要一段时间。当仿真结束时，将会显示出系统的输出，同时，参考信号也会显示出来，如图 10-31 所示。

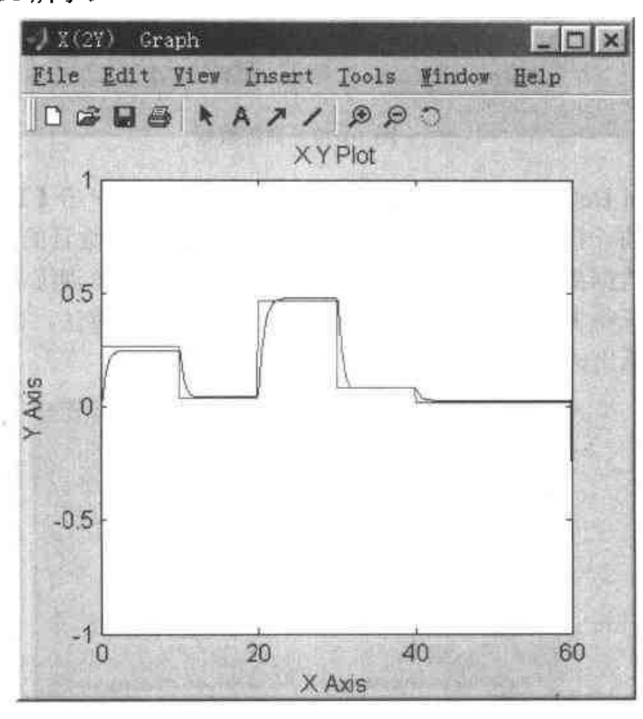


图 10-31 仿真得到的系统输出及参考信号

10.7 总 结

首先，对网络的导出与导入加以简单说明。对于本章介绍的三种神经网络控制结构，可以将设计好的网络和训练数据保存到工作空间中或是保存到磁盘文件中。这些操作将在第 11 章《图形用户接口 GUI》中进行介绍。有兴趣的读者可以参考第 11 章的相关部分，只要稍做变动即可。

下面对本章介绍的三种神经网络控制器加以总结。

- 神经网络预测控制是使用神经网络系统模型来预测系统未来的行为。优化算法用于确定控制输入，这个控制输入优化了系统在一个有限时间段里的性能。系统训练仅仅需要对于静态网络的成批训练算法，当然，训练速度非常快。控制器需要在线的优化算法，这就需要比其他控制器更多的计算。
- 对于 NARMA-L2（反馈线性化）控制，系统的近似模型转换成相伴型。计算出的下一个控制输入被用于迫使系统输出跟踪参考信号。这种神经网络系统模型使用静态反传算法，因此，速度很快。控制器是对系统模型的重整，它需要最小的在线计算。
- 对于模型参考控制，首先建立一个神经网络系统模型。接着，使用这个系统模型来训练一个神经网络控制器，迫使系统输出跟踪参考模型的输出。这种控制结构需要使用动态反传算法来训练控制器。这样，通常情况下比使用标准的反传算法训练静态网络花费的时间要多。然而，这种方法比 NARMA-L2（反馈线性化）控制结构更能适应一般的情况。这种控制器需要最少的在线计算时间。

第11章 图形用户接口GUI

与以前的版本不同，MATLAB 6.5提供的神经网络工具箱NN Toolbox 4.0.2新增了一个非常流行的功能：图形用户接口（GUI），这使得神经网络的设计变得非常容易上手，其界面也非常友好。

本章主要内容包括：

- 建立网络
- 训练网络
- 将数据导出到命令行工作空间中
- 清除数据
- 从命令行工作空间中导入数据
- 变量存盘与读取

11.1 引　言

与以前的版本不同，MATLAB 6.5提供的神经网络工具箱NN Toolbox 4.0.2新增了一个非常流行的功能：图形用户接口（GUI），这使得神经网络的设计变得非常容易上手，其界面也非常友好。

在使用图形用户接口时，将产生一个Network/Data Manager窗口。这个窗口有它自己的工作空间，这个工作空间是与我们以前经常提到的MATLAB命令行工作空间分开的。于是，在使用图形用户接口时，就有可能将其产生的结果“导出”到MATLAB命令行工作空间中。同样地，也有可能将MATLAB命令行工作空间的结果“导入”到图形用户接口工作空间中。

当Network/Data Manager窗口处于激活状态（在运行）时，就可以使用其生成一个神经网络，并且能够看到这个网络，对它进行训练、仿真，甚至可以将最后得到的结果导出到MATLAB命令行工作空间中。当然，同样可以将MATLAB命令行工作空间中的数据导入到图形用户接口中使用。

下面使用一个非常简单的例子，来说明图形用户界面的使用。将设计一个感知器神经网络，并将尽可能地使用能够用到的所有步骤，通过这些步骤，解决在设计过程中可能遇到的问题。

11.2 建立网络

在本例中，将要建立一个感知器网络，使其能够完成“与”的功能。显然，对于这个

“与”函数来说，输入向量为 p=[0 0 1 1;0 1 0 1]，目标向量为 t=[0 0 0 1]。我们给这个感知器网络命名为 ANDNet。一旦网络被建立起来，就要对其进行训练。下一步，通过将图形用户接口的数据导出到 MATLAB 命令行工作空间中，我们甚至可以完成保存网络、网络输出等工作。

11.2.1 输入和目标

开始在 MATLAB 命令行中输入 nntool，就会出现一个窗口，如图 11-1 所示，这就是前面提到的 Network/Data Manager 窗口。

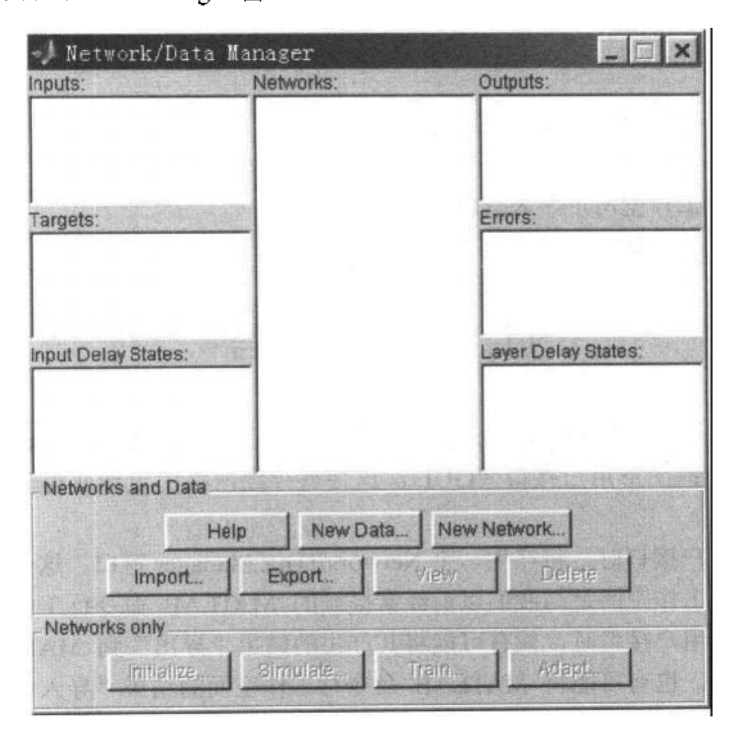


图 11-1 Network/Data Manager 窗口

观察此窗口，可见其有 7 个显示区域和若干个按钮。其中 Inputs 区域中将显示指定的输入向量；Targets 区域中将显示指定的目标向量；Input Dalay States 区域中将显示设置的输入延迟参数；Networks 区域中将显示设置的网络类型；Outputs 区域中将显示网络对应的输出值；Errors 区域中将显示误差；Layer Dalay States 区域中将显示层的延迟状态。在后面将会对它们详细介绍。

此时，如果对图形用户接口不太熟悉，可以单击【Help】按钮。这时将弹出一个新的窗口，此窗口首先介绍了使用图形用户接口解决问题的一般步骤，然后给出了关于这些按钮的描述，最后，对于每个区域各代表什么意思也做了介绍。

首先，要定义网络的输入值。令输入向量为[0 0 1 1;0 1 0 1]，记为 p。于是，网络的输入为两元素的向量，一共有 4 个这样的两元素输入向量供网络训练用。要在图形用户界面中实现这些数据的定义，只需单击【New Data】按钮。此时，将弹出一个名为 Create New

Data 的新窗口。将此窗口中 Name 区域的值设置为 p，将 Value 区域的值设置为[0 0 1 1;0 1 0 1]，并且确保右边单选框中的 Inputs 选项被选中。于是，我们将得到设置好了的输入向量设置，实际窗口如图 11-2 所示。

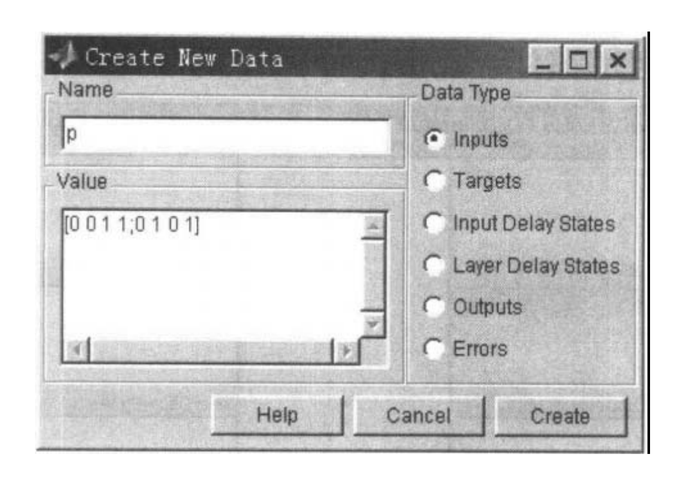


图 11-2 在 Create New Data 窗口中设置输入值

在设置好以上输入向量之后，单击【Create】按钮，就完成了在图形用户接口中输入初始向量 p 的过程。此时，返回到 Network/Data Manager 窗口中，并在其中的 Inputs 区域中将 p 作为输入显示出来。

此时，单击 Inputs 区域中刚才设置的 p 值，Network/Data Manager 窗口中的【View】按钮和【Delete】按钮被激活，说明可以显示输入 p 的值，也可以将其删除。单击【View】按钮，得到一个新的窗口，如图 11-3 所示。

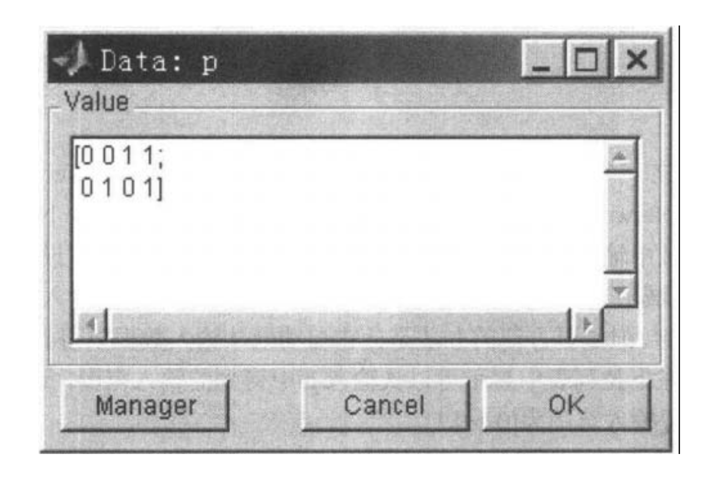


图 11-3 输入向量 p 的显示窗口

在这个窗口中显示了输入向量 p 的值。单击【Manager】按钮，控制的焦点就会回到 Network/Data Manager 窗口，可以进行其他操作。

在 Network/Data Manager 窗口中单击【Delele】按钮，则会将刚才输入的向量 p 从图

形用户接口工作空间中删除。

下面在图形用户接口中定义目标值。再次单击【New Data】按钮，在变量名区域中输入 *t*，将 Value 区域的值设置为[0 0 0 1]，并且将右边单选框中的 Targets 选项选中。在设置好以上目标向量之后，再次单击【Create】按钮，这时，就会在 Network/Data Manager 窗口的 Targets 区域中得到目标值 *t*，这与前面的 *p* 的情况相似，如图 11-4 所示。

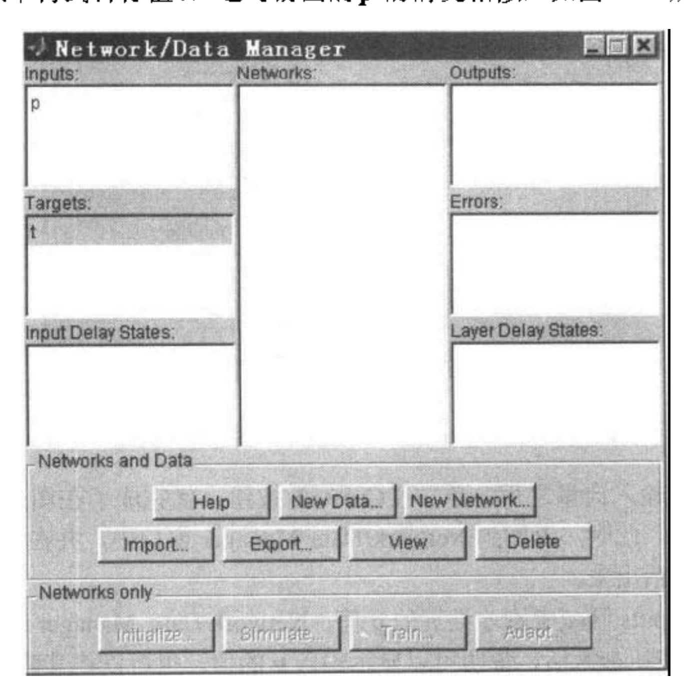


图 11-4　定义了输入与目标数据的 Network/Data Manager 窗口

11.2.2　建立网络

下一步准备建立一个新的网络，并命名为 ANDNet。首先，在 Network/Data Manager 窗口中单击【New Network】按钮，这时将弹出一个新的窗口：Create New Network。在其 Network Name 区域中输入 ANDNet 作为网络名。然后，将网络类型设置为感知器，即在其 Network Type 区域中选择 Perceptron。Input ranges 区域用于设置输入向量的范围，可以直接在其中填写数字，但是更方便的办法是让它自动地从输入数据中得到。单击 Input ranges 右边的下拉箭头，在下拉列表中显示可以从输入 *p* 中得到的输入范围。这正是我们需要的。单击 p，于是就设置输入范围为[0 1;0 1]。

下面指定传递函数和学习函数。希望使用 hardlim 函数作为传递函数，使用 learnp 函数作为学习函数。于是单击 Transfer function 右边的下拉箭头，得到下拉列表，从中选择函数 HARDLIM，单击 Transfer function 右边的下拉箭头，得到下拉列表，从中选择函数 LEARNP。

经过上述的设置后，Create New Network 窗口如图 11-5 所示。

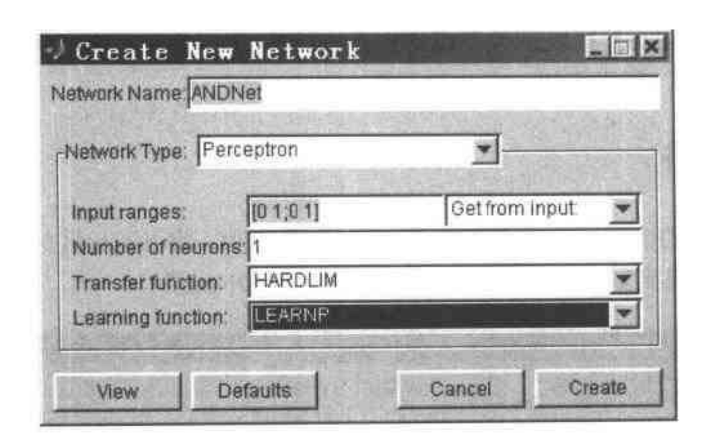


图 11-5 设置好以后的 Create New Network 窗口

在上述窗口中单击【View】按钮，可以看到建立的网络结构图，如图 11-6 所示。

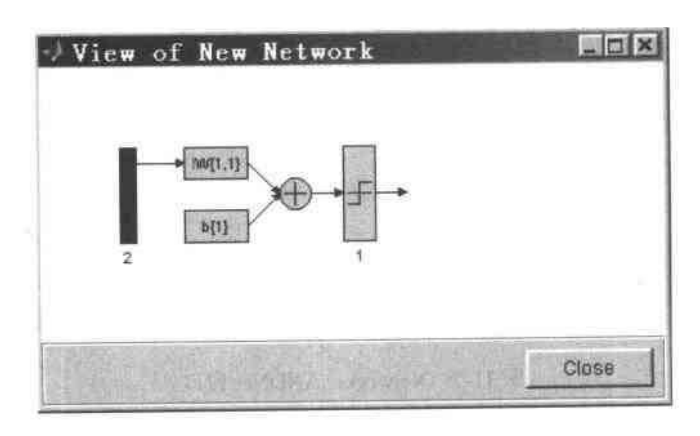


图 11-6 建立的网络结构图

从图 11-6 可以看出，建立的网络为单输入（此输入有两个分量），传递函数为 hardlim 函数，输出端也为单输出。这正是我们需要的感知器网络。

单击 Create New Network 窗口中的【Create】按钮，就会在图形用户接口工作空间中建立起这个网络。此时，又会回到 Network/Data Manager 窗口的控制中，可以发现，ANDNet 出现在 Networks 区域中。

至此，已经在图形用户接口中建立起了需要的网络。

11.3 训 练 网 络

下面训练刚才建立的网络。首先，在 Network/Data Manager 窗口中单击 ANDNet，将其激活。这时，可以发现 Networks only 区域中的四个按钮【Initialize】、【Simulate】、【Train】、【Adapt】变为有效。单击【Train】按钮，将弹出一个新的窗口，名为 Network: ANDNet。在这个窗口中，单击左上角的【View】按钮，也能够看到网络的结构图。此时，

还可以单击【Initialize】按钮来改变或者检查网络的初始值。这些操作都基本类似，读者可以试着自己完成，限于篇幅，在此就不赘述了。

11.3.1 训练网络

在选中 Train 的情况下，再选中 Training Info 子选项，将 Inputs 区域选为 p，将 Targets 区域选为 t。此时得到如图 11-7 所示的 Network：ANDNet 窗口。

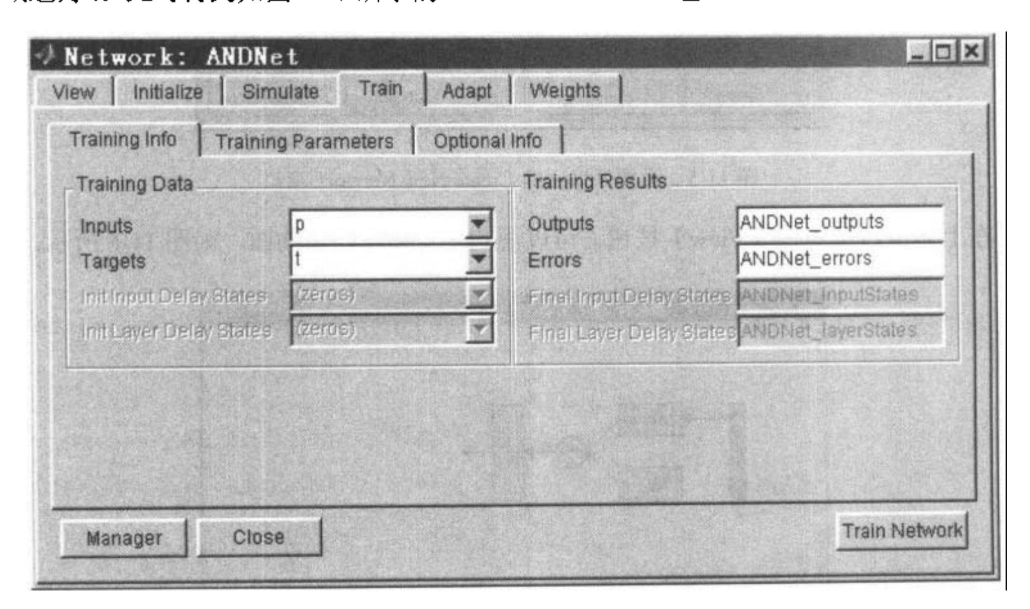


图 11-7　Network：ANDNet 窗口



在 Training Results 中，Outputs 和 Errors 区域中的变量 ANDNet_outputs 及 ANDNet_errors 命名中都在前面附上了“ANDNet”。这样的命名使得当它们被导出到 MATLAB 命令行工作空间中时能很容易地辨别出来。

在 Network：ANDNet 窗口中单击 Training Parameters，将显示出一些训练参数。例如：训练步数、误差目标等。同样，可以在这里更改这些训练参数。

在 Network：ANDNet 窗口中单击 Optional Info，将显示出另外一些设置，同样，可以在这里根据需要改变这些参数。

这时，单击 Network：ANDNet 窗口右下方的【Train Network】按钮，就可以开始网络的训练。训练过程如图 11-8 所示。

从图 11-8 中可以看出，网络在 6 个时间步长里就完成了训练，误差变为了 0。



这是对感知器网络而言的，对于其他网络来说，通常是不能将误差变为 0 的，甚至误差范围还比较大。在那种情况下，就不能像在这里一样在线性坐标图上绘出训练过程，而是在对数坐标图上绘出。

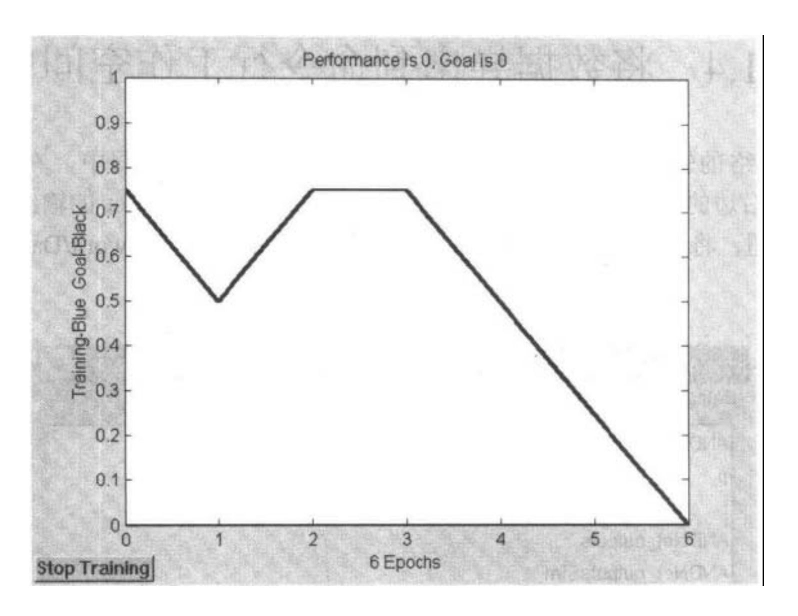


图 11-8 训练过程

11.3.2 仿真网络

下面使用输入 p 来仿真网络，测试其经过训练后是否能够对于输入 p 以 0 误差输出。回到 Network/Data Manager 窗口，单击【Network only：Simulate】按钮，弹出窗口 Network：ANDNet，再单击 Simulate。在 Inputs 右边的下拉列表中选择 p 作为输入，同时将 Outputs 栏中的变量名改为 ANDNet_outputsSim，以便与训练的输出相区分。单击窗口右下角的【Simulate Network】按钮，回到 Network/Data Manager 窗口，将会看到在 Outputs 区域中有一个新的变量 ANDNet_outputsSim。双击这个变量，将会弹出一个新的窗口 Data：ANDNet_outputsSim，如图 11-9 所示。

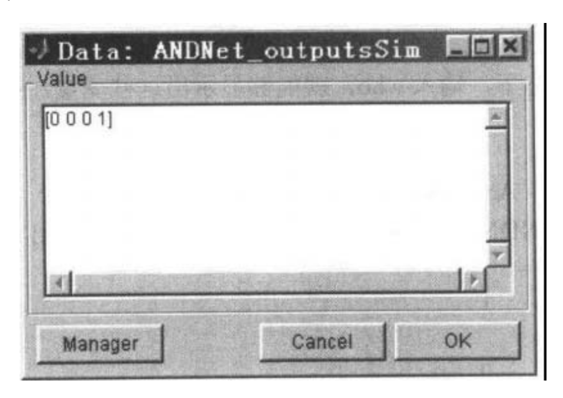


图 11-9 仿真输出

从图 11-9 中可以看出，网络的仿真输出结果正好实现了“与”的功能。

11.4 将数据导出到命令行工作空间中

可以将网络的输出及误差导出到 MATLAB 命令行工作空间中。在 Network/Data Manager 窗口右边的 Outputs 和 Errors 区域中可以看到网络 ANDNet 的输出和误差。单击【Export】按钮，将会弹出一个新的窗口：Export or Save from Network/Data Manager，如图 11-10 所示。

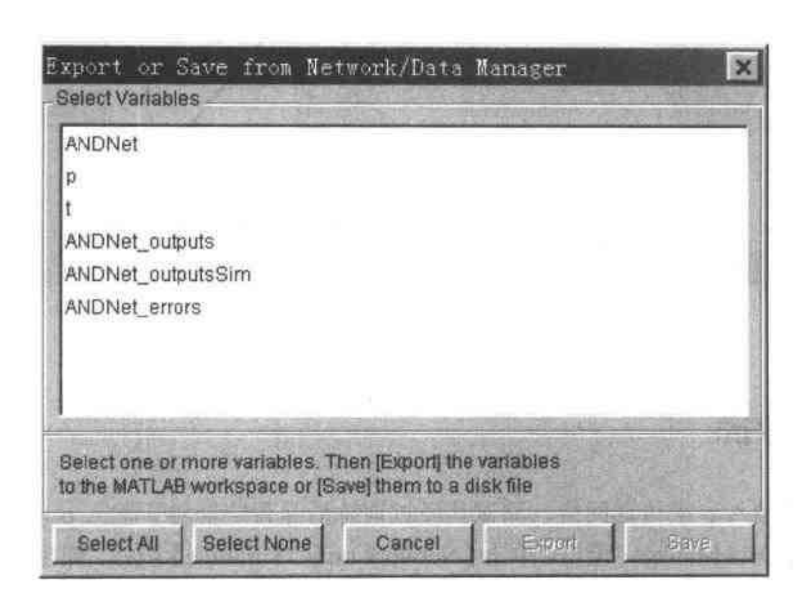


图 11-10 导出或保存窗口

在这个窗口中，列出了刚才建立的网络及相关的变量。可以在其中选择需要导出到 MATLAB 命令行工作空间的变量。

单击 ANDNet_outputs 及 ANDNet_errors，使它们被选中，然后单击【Export】按钮。这两个变量就会在 MATLAB 命令行工作空间中出现。这可以通过下面的方法来验证：回到 MATLAB 命令行状态下，输入 who，就能够显示出这些变量，结果如下：

```
who
Your variables are:
ANDNet_errors    ANDNet_outputs
```

可以进一步输入 ANDNet_outputs 和 ANDNet_errors 来显示出它们的值。

```
ANDNet_outputs =
0       0       0       1
ANDNet_errors =
0       0       0       0
```

使用同样的方法，将变量 p、t 和 ANDNet 导出到 MATLAB 命令行工作空间中。同样也可以使用 who 来显示出这些变量。

```
who
Your variables are:
```

```
ANDNet    ANDNet_errors    ANDNet_outputs    p    t
```

既然网络 ANDNet 已经被导出到了 MATLAB 命令行工作空间中，那么，就可以得到网络的权重矩阵。输入命令：

```
ANDNet.iw{1,1}
```

于是可以得到权重为：

```
ans =
1
```

同样，可以得到网络的阈值，输入命令：

```
ANDNet.b{1}
```

可以得到阈值为：

```
ans =
-3
```

11.5 清除数据

有时，需要将图形用户接口中的某些数据清除掉。可以在 Network/Data Manager 窗口中单击需要清除的变量，使其被激活，然后单击窗口右下方的【Delete】按钮，这个变量就会被删除。逐个选中变量，并把它们都删除掉，这样就可以清除图形用户界面工作空间中的数据。

当然，更快的办法是退出 MATLAB，然后重新进入，重新在命令行中输入 nntool，进入图形用户界面。



清除图形用户界面工作空间的数据并不影响到那些已经被导出到 MATLAB 命令行工作空间的数据。即使是关闭了 Network/Data Manager 窗口，它们还是在命令行工作空间中存在。

11.6 从命令行工作空间中导入数据

前面介绍了将数据导出到命令行工作空间的方法，现在讨论将数据从命令行工作空间中导入图形用户接口工作空间的方法。

为了简化步骤，我们退出 MATLAB，然后重新进入。在命令行中输入 nntool 重新进入图形用户接口。

在 MATLAB 命令行中定义一个变量：

```
r=[0;1;2;3]
r =
0
1
2
3
```

回到 Network/Data Manager 窗口，单击【Import】按钮，将弹出一个新的窗口：Import or Load to Network/Data Manager。设置 Source 为 Import from MATLAB workspace，并选择变量 r，设置目标变量名也为 r，需要注意的是，这两个 r 处于不同的工作空间中，代表着不同的变量。在 Import 区域中选中 Inputs 选项。这时得到如图 11-11 所示的窗口。

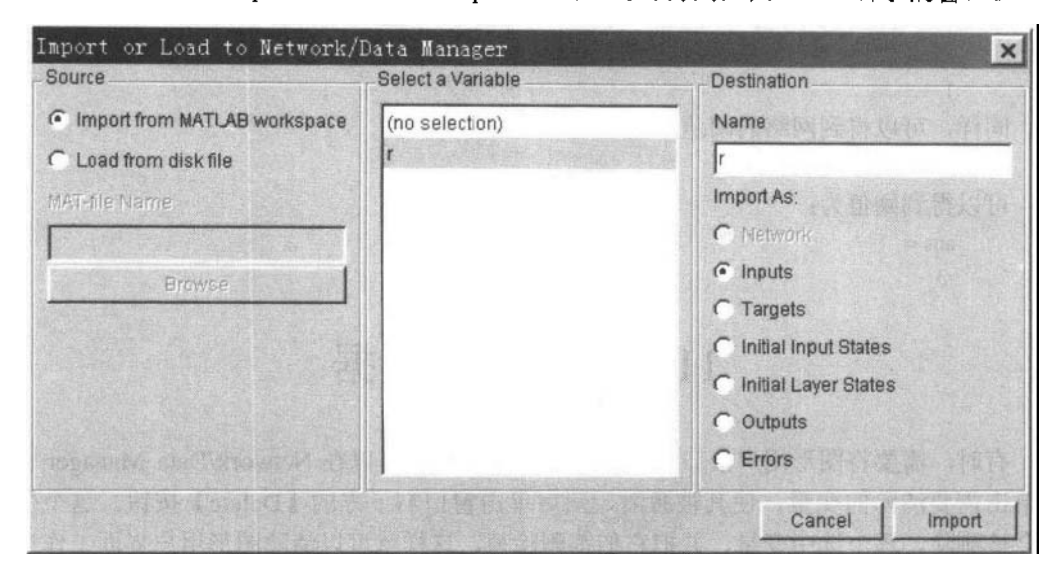


图 11-11　到入数据窗口

单击此窗口的【Import】按钮，就将此数据导入到了图形用户接口工作空间中。此时，在 Network/Data Manager 窗口中能够看到新导入的数据 r。



在图 11-11 显示的 Import or Load to Network/Data Manager 窗口中，有一个选项 Load from disk file，下面可以指定地址。这说明不但可以从命令行工作空间中导入数据，还可以从存盘文件中导入数据。这项操作将在下一节介绍。

11.7　变量存盘与读取

在图形用户接口工作空间中定义的变量可以以 MAT 文件的形式保存下来，以便于在以后需要处理时，能够再将其读出。下面简单地介绍一下这一过程。

首先进入 Network/Data Manager 窗口，建立一个新的网络，这里只需单击【New Network】按钮。将这个新的网络命名为 mynet，然后单击【Create】按钮，于是网络名 mynet 将会出现在 Network/Data Manager 窗口中。

下面保存这个网络。单击【Export】按钮，将会弹出一个新的窗口：Export or Save from Network/Data Manager。在这个窗口的变量列表中选择 mynet，并且单击窗口右下角的【Save】按钮。

此时，保存窗口弹出，选择保存路径和文件名（mynetfile），就可以将选定的网络保

第 12 章　SIMULINK

神经网络工具箱提供了一套可以在 SIMULINK 中用来建立网络的模块，对于在 MATLAB 工作空间中建立的网络，也能够使用函数 gensim 来生成一个相应的 SIMULINK 版的网络。

12.1　模块的设置

在 MATLAB 命令行中输入以下命令，就能够得到神经网络工具箱的模块窗口：Library：neural，如图 12-1 所示。

Neural

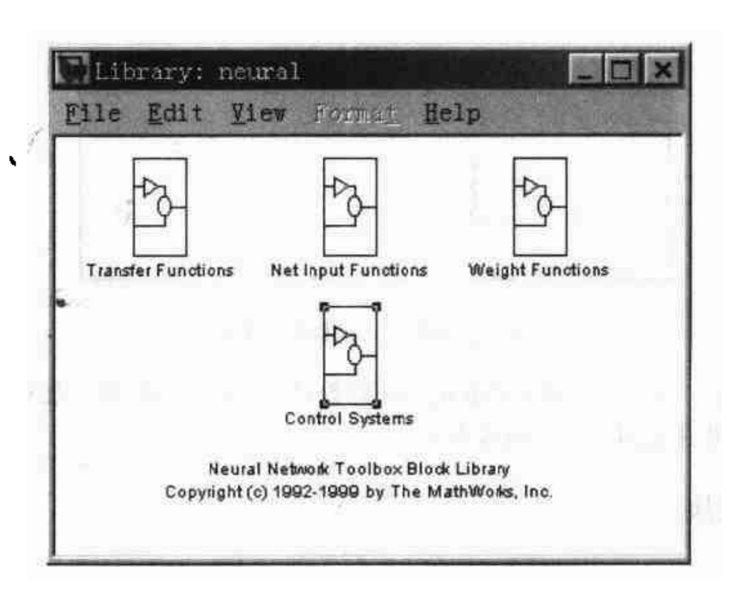


图 12-1　神经网络工具箱组件窗口

从图 12-1 可以看出，神经网络工具箱中包含了四个模块。其实，在 MATLAB 5.3 对应的神经网络工具箱 NN Toolbox 3.0 中只有三个模块，而 MATLAB 6.5 对应的神经网络工具箱 NN Toolbox 4.0.2 增加了一个模块：Control Systems。

对于上面的四个模块，每一个又都包含了一些附加的模块。

12.1.1　传递函数模块

双击 Library：neural 窗口中的 Transfer Functions 模块，将弹出一个新窗口：Library：neural/Transfer Functions，其中包含了一些传递函数模块，如图 12-2 所示。

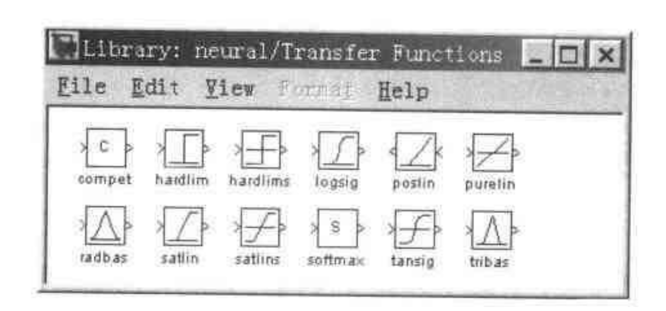


图 12-2 传递函数模块窗口

在图 12-2 中，每一个模块都能够接受一个网络输入向量，并且相应地产生一个输出向量，这个输出向量的维数和输入向量相同。

12.1.2 网络输入模块

双击 Library：neural 窗口中的 Net Input Functions 模块，将弹出一个新窗口：Library：neural/Net Input Functions，其中包含了两个网络输入函数模块，如图 12-3 所示。

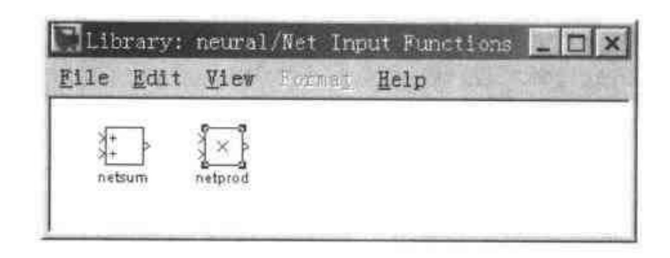


图 12-3 网络输入函数模块窗口

在图 12-3 中，每一个模块都能够接受任意数目的加权输入向量、加权的层输出向量，以及阈值向量，并且返回一个网络输入向量。

12.1.3 权重模块

双击 Library：neural 窗口中的 Weight Functions 模块，将弹出一个新窗口：Library：neural/Weight Functions，其中包含了四个权重函数模块，如图 12-4 所示。

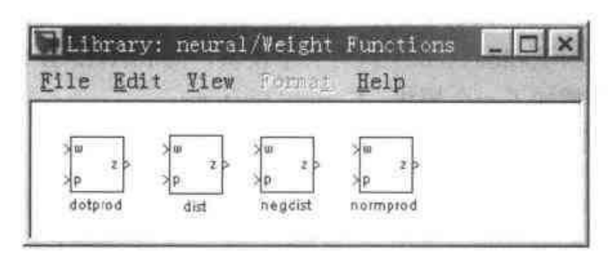


图 12-4 权重函数模块窗口

在图 12-4 中，每个模块都以一个神经元权中向量作为输入，并将其与一个输入向量（或者是某一层的输出向量）进行运算，得到神经元的加权输入值。



上面的这些模块需要的权重向量必须定义为列向量。这是因为 SIMULINK 中的信号可以为列向量，但是不能为矩阵或者行向量。

另外，正是由于上述的原因，在将一个权重矩阵加到一个具有 *S* 个神经元的网络层上时，必须生成 *S* 个权重函数组件。

12.1.4 控制系统模块

双击 Library: neural 窗口中的 Control Systems 模块，将弹出一个新窗口：Library: neural/Control Systems，其中包含了四个控制系统模块，如图 12-5 所示。

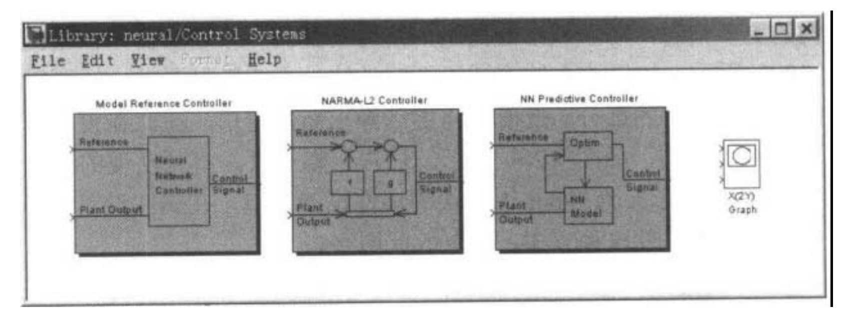


图 12-5 控制系统模块

控制系统模块是 MATLAB 6.5 对应的神经网络工具箱 NN Toolbox 4.0.2 新增加的内容，使得神经网络控制的 MATLAB 设计变得更加容易。

图 12-5 中的四个控制系统模块中，有三个是控制器，剩下的一个是示波器。关于在神经网络中这三个控制器的使用，将在下一节专门介绍。

12.2 模块的生成

使用函数 gensim 能够在 MATLAB 命令行中对一个网络生成其模块化描述，这使得我们能够在 SIMULINK 中对网络进行仿真。

```
gensim(net,st)
```

gensim 函数的第一个参数指定了需要生成模块化描述的网络，第二个参数指定了采样时间，它通常情况下为某个正的实数。

如果网络没有与输入权重或者层中权重相关的延迟，则指定第二个参数为−1。一旦第二个参数为−1，那么函数 gensim 将生成一个连续采样的网络。

下面举一个简单的例子来介绍其用法。

首先，在 MATLAB 命令行中定义网络的输入 p 和相应的目标 t。

```
p=[1 2 3 4 5];
```

```
t=[1 3 5 7 9];
```

然后，使用函数 newlind 来设计一个线性网络来解决这个问题。

```
net=newlind(p,t)
```

接下来，使用函数 sim 来测试网络，输入数据为原始的数据。

```
y=sim(net,p)
```

在 MATLAB 命令行中将得到如下的结果。可以看出，网络已经正确地解决了问题。

```
y =
1.0000    3.0000    5.0000    7.0000    9.0000
```

调用函数 gensim 生成上述网络的 SIMULINK 版本。

```
Gensim(net,-1)
```

上述命令中的第二个参数－1 指定了将生成一个连续采样的网络模块。

调用函数 gensim 后，将生成一个新的窗口。在这个窗口中包含有一个线性网络的仿真，这个线性网络连接上了一个采样输入和一个示波器，如图 12-6 所示。

其中线性网络值使用一个模块代替，并没有具体地显示出其网络结构。如果想得到关于此网络结构的更多信息，可以双击此网络，这时将弹出一个新的窗口，绘出了此网络的结构，如图 12-7 所示。如果这个结构图还不够具体，不能满足要求，还可以进一步在其基础上双击需要详细了解的部分。此时，将继续弹出新的窗口，在此新窗口中出现了刚才单击部分的更详细的结构，如图 12-8 所示。这样一直下去，直到出现的窗口为属性设置窗口为止，如图 12-9 所示即为 Input 1 模块的属性设置窗口。在属性设置窗口中，可以改变网络的某些属性值，通过这种方法，来更改此网络。

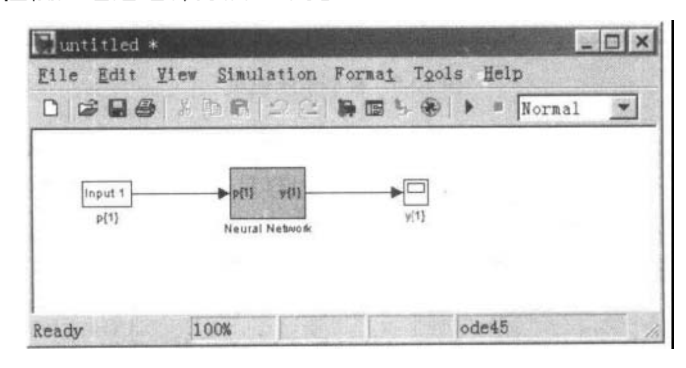


图 12-6 使用 gensim 生成的网络模块

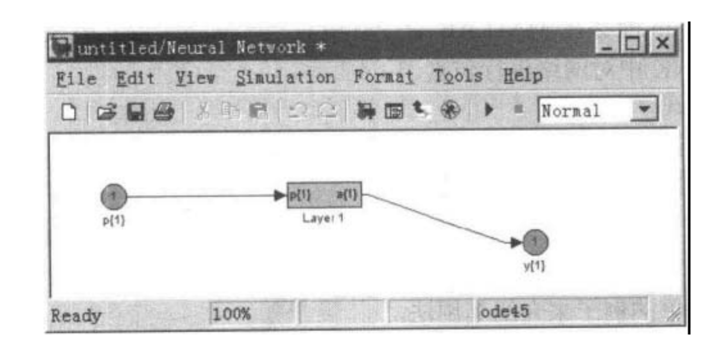


图 12-7 双击网络后得到的网络结构图

图 12-8　双击网络层后得到的详细网络结构

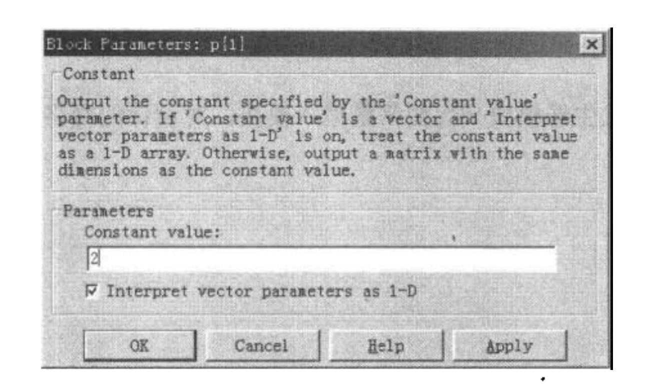


图 12-9　Input 1 模块的属性设置窗口

网络的 Input 1 模块实际上是一个标准的常量模块。在图 12-9 所示的窗口中将 Constant value 区域的值由初始时的随机值改为 2，然后单击【OK】按钮。

回到图 12-6 所示的窗口，在【Simulation】菜单下选择【Start】菜单项，便开始在 SIMULINK 中对网络进行仿真。

为了观察结果，可以双击窗口中的示波器 y{1}，此时，将弹出一个新的窗口，显示了输出信号的波形，如图 12-10 所示。

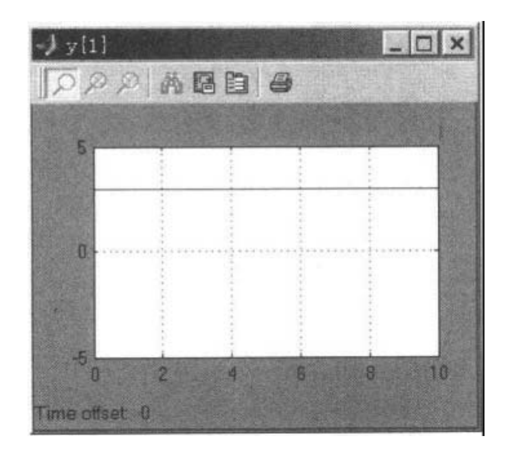


图 12-10　输出信号的波形

从图 12-10 中可以看出，输出为 3，这与我们在 MATLAB 命令行中使用函数 sim 得到的结果是一致的。

下面还有两项工作没有完成，由于这些操作基本类似，本处就不再赘述了，有兴趣的读者可以试着自己完成。

1. 改变输入信号

在标准 SIMULINK 的 Source 模块中选择一个信号发生器模块，用来代替本例中的常量输入模块。重新仿真这个新的网络，观察网络的响应。

2. 离散的采样时间

重新建立网络，这次使用离散采样时间 0.5，而不再使用前面的连续采样时间，于是有下面的命令：

```
gensim(net,0.5)
```

再一次使用信号发生器模块来代替常量输入模块，对设计的网络进行仿真，观察网络的响应。

第13章　高 级 话 题

一般情况下，在实际应用中，使用前面介绍的 MATLAB 工具箱函数就足以设计出所需的神经网络。但是，对于某些特定的需要，往往希望定制网络，或者希望得到更多的功能，或者希望定制函数。本章就基于这些需求介绍这些内容。

本章主要讲述以下内容：

- 定制网络
- 附加的工具箱函数
- 定制函数

13.1　定 制 网 络

神经网络工具箱被设计成允许使用多种网络，这使得很多函数能使用相同的网络对象数据类型。所有的标准工具箱网络生成函数如表 13-1 所示。

表 13-1　标准工具箱网络生成函数

函数名称	功　能
newc	生成一个竞争层
newcf	生成一个前向层叠 BP 网络
newlm	生成一个 Elman BP 网络
newff	生成一个前馈 BP 网络
newfftd	生成一个前馈输入延时 BP 网络
newgrnn	设计一个泛化的回归网络
newhop	生成一个 Hopfield 回归网络
newlin	生成一个线性网络
newlind	设计一个线件网络
newlvq	生成一个学习向量量化网络
newp	生成一个感知器
newpnn	设计一个概率神经网络
newrb	设计一个径向基网络
newrbe	设计一个精确的径向基网络
newsom	生成一个自组织映射网络

由于对于网络有一个面向对象的表示，因此，其灵活性是有可能得到保证的。这种表示允许定义不同的结构，同时，对这些结构也允许使用不同的算法。

我们从一个空的网络开始来生成一个定制的网络（通过 network 函数来实现），将其属性设置为所需的属性。

network 生成一个定制的神经网络

网络对象包含很多属性，我们可以设置这些属性，使其适合所设计的网络结构及作用。下面的部分演示了怎样使用这些属性生成一个定制的网络。

13.1.1 定制网络

在建立一个网络之前，需要对其有所了解。本小节将介绍生成一个复杂的网络的方法，其结构如图 13-1 所示。

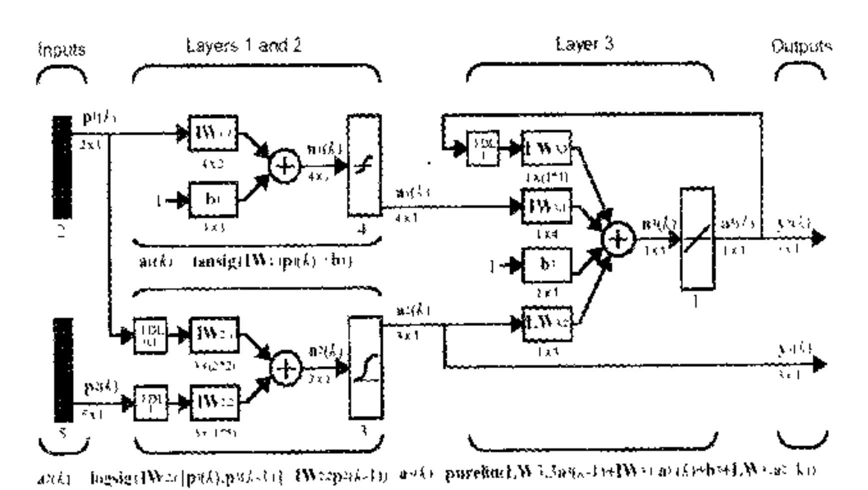


图 13-1 网络结构

在这个网络中，对于第一个网络输入的两个元素，每一个的允许输入值都在 0~10 之间；对于第二个网络输入的五个元素，每一个的允许输入值在−2~2 之间。

在完成设计之前，首先需要指定其使用的初始化，以及训练算法。

假设每一层的权重及偏置都使用 Nguyen-Widrow 层初始化方法（initnw）来进行初始化并且网络使用 Levenberg-Marquardt 回传方法（trainlm）进行训练，这样，给定一些例子作为输入向量，第三层输出将学习，并以误差均方值最小（mse）来实现对目标向量进行匹配。

13.1.2 网络定义

首先要生成一个新的网络。输入下面的代码，生成一个新的网络，并且观察它的很多属性。

```
net = network
```

可以得到下面的输出显示：

```
net =
Neural Network object:
```

```
architecture:
numInputs: 0
numLayers: 0
biasConnect: []
inputConnect: []
layerConnect: []
outputConnect: []
targetConnect: []
numOutputs: 0    (read-only)
numTargets: 0    (read-only)
numInputDelays: 0    (read-only)
numLayerDelays: 0    (read-only)
subobject structures:
inputs: {0x1 cell} of inputs
layers: {0x1 cell} of layers
outputs: {1x0 cell} containing no outputs
targets: {1x0 cell} containing no targets
biases: {0x1 cell} containing no biases
inputWeights: {0x0 cell} containing no input weights
layerWeights: {0x0 cell} containing no layer weights
functions:
adaptFcn: (none)
initFcn: (none)
performFcn: (none)
trainFcn: (none)
parameters:
adaptParam: (none)
initParam: (none)
performParam: (none)
trainParam: (none)
weight and bias values:
IW: {0x0 cell} containing no input weight matrices
LW: {0x0 cell} containing no layer weight matrices
b: {0x1 cell} containing no bias vectors
other:
userdata: (user stuff)
```

1．结构属性

显示的第一组属性标识为结构属性。这些属性允许选择输入，以及层的数目，也允许选择它们的连接。

1）输入及层的数目

显示的前两个属性是 numInputs 和 numLayers。这两个属性允许根据网络需要选择输入的数目，以及层的数目。

```
net =
Neural Network object:
architecture:
```

```
numInputs: 0
numLayers: 0
```

可以看出，现在网络没有输入和层。

根据定制网络的需要，将这两个属性值改为需要的输入及层的数目。

```
net.numInputs = 2;
net.numLayers = 3;
```

需要注意，net.numInputs 是输入源的数目，而不是在一个输入向量中的元素的数目（net.inputs{i}.size）。

2）偏置连接

输入 net，按回车键，再次观察其属性。现在网络有两个输入和三层。

```
net =
Neural Network object:
architecture:
numInputs: 2
numLayers: 3
biasConnect: [0; 0; 0]
inputConnect: [0 0; 0 0; 0 0]
layerConnect: [0 0 0; 0 0 0; 0 0 0]
outputConnect: [0 0 0]
targetConnect: [0 0 0]
numOutputs: 0   (read-only)
numTargets: 0   (read-only)
numInputDelays: 0   (read-only)
numLayerDelays: 0   (read-only)
subobject structures:
inputs: {2x1 cell} of inputs
layers: {3x1 cell} of layers
outputs: {1x3 cell} containing no outputs
targets: {1x3 cell} containing no targets
biases: {3x1 cell} containing no biases
inputWeights: {3x2 cell} containing no input weights
layerWeights: {3x3 cell} containing no layer weights
functions:
adaptFcn: (none)
initFcn: (none)
performFcn: (none)
trainFcn: (none)
parameters:
adaptParam: (none)
initParam: (none)
performParam: (none)
trainParam: (none)
weight and bias values:
IW: {3x2 cell} containing no input weight matrices
LW: {3x3 cell} containing no layer weight matrices
```

```
b: {3x1 cell} containing no bias vectors
other:
userdata: (user stuff)
```

注意接下来的五个属性：

```
biasConnect: [0; 0; 0]
inputConnect: [0 0; 0 0; 0 0]
layerConnect: [0 0 0; 0 0 0; 0 0 0]
outputConnect: [0 0 0]
targetConnect: [0 0 0]
```

这些由 0 和 1 组成的矩阵表示了偏置、输入权重、层权重、输出，以及目标连接是否存在。它们现在都是 0，这意味着网络没有任何这样的连接。

注意到偏置连接矩阵是一个 3 乘 1 的向量。为了对第 i 层产生一个偏置连接，可以设置 net.biasConnect(i)为 1。根据我们的需要，第一层和第三层具有偏置连接，可以如下所示输入这些属性：

```
net.biasConnect(1) = 1;
net.biasConnect(3) = 1;
```

其实也可以用一行代码来完成这些连接的定义，如下所示：

```
net.biasConnect = [1; 0; 1];
```

3）输入及层的权重连接

输入连接矩阵是一个 3 乘 2 的矩阵，它表示从两个源端（两个输入）到三个目的端（三个层）之间的连接。这样，net.inputConnect(i,j)就表示了从第 j 个输入到第 i 层之间的一个权重连接。

为了连接第一个输入到第一层，以及第二个输入到第二层（如前面图 13-1 中所示的情形），可输入下面代码：

```
net.inputConnect(1,1)=1;
net.inputConnect(2,1)=1;
net.inputConnect(2,2)=1;
```

或者输入下面的一行代码：

```
net.inputConnect=[1 0; 1 1; 0 0];
```

类似地，net.layerConnect(i,j)表示了一个从第 j 层连接到第 i 层的层权重连接。按照如下的方式连接第 1、2 层，以及第 3 层到第 4 层。

```
net.layerConnect=[0 0 0; 0 0 0; 1 1 1];
```

4）输出及目标连接

输出和目标连接矩阵都是 1 乘 3 的矩阵，这意味着它们从三个源端（三个层）连接到一个目标（外部世界）。

连接第 2 层和第 3 层到网络输出，输入下面的代码：

```
net.outputConnect=[0 1 1];
```

给定第 3 层一个目标连接，输入下面的代码：

```
net.targetConnect=[0 0 1];
```

第 3 层目标与第 3 层输出比较，产生一个误差信号，当测量网络性能或者是在训练、调整中更新网络时，需要使用这个误差信号。

现在，输入 net 命令，再次观察网络的属性值：

```
net =
Neural Network object:
architecture:
numInputs: 2
numLayers: 3
biasConnect: [1; 0; 1]
inputConnect: [1 0; 1 1; 0 0]
layerConnect: [0 0 0; 0 0 0; 1 1 1]
outputConnect: [0 1 1]
targetConnect: [0 0 1]
numOutputs: 2   (read-only)
numTargets: 1   (read-only)
numInputDelays: 0   (read-only)
numLayerDelays: 0   (read-only)
subobject structures:
inputs: {2x1 cell} of inputs
layers: {3x1 cell} of layers
outputs: {1x3 cell} containing 2 outputs
targets: {1x3 cell} containing 1 target
biases: {3x1 cell} containing 2 biases
inputWeights: {3x2 cell} containing 3 input weights
layerWeights: {3x3 cell} containing 3 layer weights
functions:
adaptFcn: (none)
initFcn: (none)
performFcn: (none)
trainFcn: (none)
parameters:
adaptParam: (none)
initParam: (none)
performParam: (none)
trainParam: (none)
weight and bias values:
IW: {3x2 cell} containing 3 input weight matrices
LW: {3x3 cell} containing 3 layer weight matrices
b: {3x1 cell} containing 2 bias vectors
other:
userdata: (user stuff)
```

可以看到，相应的属性值发生了变化。网络的偏置连接、输入连接、层连接、输出连接、目标连接均为设定值。网络输出的数目为 2，网络目标数目为 1。

2. 输出及目标数目

输入 net 并且按回车键，可以看到这些更新了的属性。其中，最后四个结构属性是只读的，这就意味着它们的值是由我们选择的其他属性的值来确定的。前面两个只读属性具有如下值：

```
numOutputs: 2   (read-only)
```

```
numTargets: 1   (read-only)
```

通过定义来自第2层和第3层的输出连接，以及来自第3层的目标连接，可以指定网络具有两个输出和一个目标。

3. 子对象属性

接下来的一组属性是：

```
subobject structures:
inputs: {2x1 cell} of inputs
layers: {3x1 cell} of layers
outputs: {1x3 cell} containing 2 outputs
targets: {1x3 cell} containing 1 target
biases: {3x1 cell} containing 2 biases
inputWeights: {3x2 cell} containing 3 input weights
layerWeights: {3x3 cell} containing 3 layer weights
```

4. 输入

在将输入数目（net.numInputs）设置为2时，输入属性变成了一个细胞数组，它具有两个输入机构。每个第 i 个输入结构（net.inputs{i}）包含另外的属性，它与第 i 个输入有关。

输入如下的代码，可以看到输入结构是怎么组织的：

```
net.inputs
```

得到如下的结果：

```
ans =
[1x1 struct]
[1x1 struct]
```

为了观察与第一个输入相关的属性，可输入如下代码：

```
net.inputs{1}
```

显示出如下的属性值：

```
ans =
range: [0 1]
size: 1
userdata: [1x1 struct]
```

注意到范围属性只有一行。这意味着输入只有一个元素，其值的范围是从0～1。size属性也指出输入只有一个元素。

定制网络的第一个输入向量有两个元素，其变化范围从0～10。用下面的方法改变这个范围属性，以达到指定的要求。

```
net.inputs{1}.range = [0 10; 0 10];
```

如果再次检查第一个输入的结构，可以看到现在它具有正确的大小，它是从新的范围值获取的。

```
ans =
range: [2x2 double]
size: 2
userdata: [1x1 struct]
```

将第二个输入向量的五个元素的范围设置为从−2～2，如下所示：

```
net.inputs{2}.range = [-2 2; -2 2; -2 2; -2 2; -2 2];
```

5. 层

当我们把层的数目（net.numLayers）设置为 3 时，层属性变成了一个三层结构的细胞数组。输入下面的代码，以观察与第一层相关的属性。

```
net.layers{1}
```

可以得到：

```
ans =
dimensions: 1
distanceFcn: ''
distances: []
initFcn: 'initwb'
netInputFcn: 'netsum'
positions: 0
size: 1
topologyFcn: 'hextop'
transferFcn: 'purelin'
userdata: [1x1 struct]
```

根据需要定制的网络的需要，输入下面三行代码，改变第一层的大小为 4 个神经元，并把其传递函数改为 tansig，把其初始化函数设为 Nguyen-Widrow 函数。

```
net.layers{1}.size = 4;
net.layers{1}.transferFcn = 'tansig';
net.layers{1}.initFcn = 'initnw';
```

第二层将具有 3 个神经元，使用 logsig 传递函数，并且用 initnw 进行初始化。这样，可以用如下所示的代码进行第二层属性的设置。

```
net.layers{2}.size = 3;
net.layers{2}.transferFcn = 'logsig';
net.layers{2}.initFcn = 'initnw';
```

第三层的大小，以及传递函数属性不需要改变，因为其所需的值正好与默认值相同。只需要设置其初始化函数，如下所示：

```
net.layers{3}.initFcn = 'initnw';
```

这时，网络相应的层属性如下所示。

第一层：

```
ans =
dimensions: 4
distanceFcn: ''
distances: []
initFcn: 'initnw'
netInputFcn: 'netsum'
positions: [0 1 2 3]
size: 4
topologyFcn: 'hextop'
transferFcn: 'tansig'
userdata: [1x1 struct]
```

第二层：

```
ans =
dimensions: 3
distanceFcn: ''
distances: []
initFcn: 'initnw'
netInputFcn: 'netsum'
positions: [0 1 2]
size: 3
topologyFcn: 'hextop'
transferFcn: 'logsig'
userdata: [1x1 struct]
```

第三层：

```
ans =
dimensions: 1
distanceFcn: ''
distances: []
initFcn: 'initnw'
netInputFcn: 'netsum'
positions: 0
size: 1
topologyFcn: 'hextop'
transferFcn: 'purelin'
userdata: [1x1 struct]
```

6．输出及目标

输入下面的代码，观察输出属性是如何组织的。

```
net.outputs
```

结果为：

```
ans =
[]     [1x1 struct]     [1x1 struct]
```

注意到输出包含了两个输出结构体，一个是第二层的，一个是第三层的。当net.outputConnect被设置为[0 1 1]时，这种组织方式就自动产生了。

用下面的代码来观察第二层的输出结构。

```
net.outputs{2}
```

得到结果如下：

```
ans =
size: 3
userdata: [1x1 struct]
```

当第二层大小（net.layers{2}.size）设置为3时，size参数自动设置为这个值。如果需要证实这一点的话，可以观察第三层的输出结构体，它同样具有正确的大小。

类似地，目标包含了一个结构体，它表示了第三层的目标。输入下面的两行代码，可以看到目标是怎样组织的，同时也可以看到第三层的目标属性。

```
net.targets
```

可以得到如下的结果：

```
ans =
[]      []      [1x1 struct]
```

输入：

```
net.targets{3}
```

得到：

```
ans =
size: 1
userdata: [1x1 struct]
```

7．偏置、输入权重，以及层权重

输入下面的代码，可以看到偏置和权重结构体是怎样组织的。

输入：

```
net.biases
```

得到：

```
ans =
[1x1 struct]
[]
[1x1 struct]
```

输入：

```
net.inputWeights
```

得到：

```
ans =
[1x1 struct]            []
[1x1 struct]      [1x1 struct]
[]                []
```

输入：

```
net.layerWeights
```

得到：

```
ans =
[]                []                []
[]                []                []
[1x1 struct]      [1x1 struct]      [1x1 struct]
```

检查这些结果，可以发现每个结果都包含了一个结构体，其所在的地方相应的连接（net.biasConnect、net.inputConnect 和 net.layerConnect）包含了一个 1。

使用下面的代码来观察它们的结构。

```
>> net.biases{1}
ans =
initFcn: ''
learn: 1
learnFcn: ''
learnParam: ''
size: 4
userdata: [1x1 struct]
```

```
>> net.biases{3}
ans =
initFcn: ''
learn: 1
learnFcn: ''
learnParam: ''
size: 1
userdata: [1x1 struct]
>> net.inputWeights{1,1}
ans =
delays: 0
initFcn: ''
learn: 1
learnFcn: ''
learnParam: ''
size: [4 2]
userdata: [1x1 struct]
weightFcn: 'dotprod'
>> net.inputWeights{2,1}
ans =
delays: 0
initFcn: ''
learn: 1
learnFcn: ''
learnParam: ''
size: [3 2]
userdata: [1x1 struct]
weightFcn: 'dotprod'
>> net.inputWeights{2,2}
ans =
delays: 0
initFcn: ''
learn: 1
learnFcn: ''
learnParam: ''
size: [3 4]
userdata: [1x1 struct]
weightFcn: 'dotprod'
>> net.layerWeights{3,1}
ans =
delays: 0
initFcn: ''
learn: 1
learnFcn: ''
learnParam: ''
size: [1 4]
```

```
userdata: [1x1 struct]
weightFcn: 'dotprod'
>> net.layerWeights{3,2}
ans =
delays: 0
initFcn: ''
learn: 1
learnFcn: ''
learnParam: '
size: [1 3]
userdata: [1x1 struct]
weightFcn: 'dotprod'
>> net.layerWeights{3,3}
ans =
delays: 0
initFcn: ''
learn: 1
learnFcn: ''
learnParam: ''
size: [1 1]
userdata: [1x1 struct]
weightFcn: 'dotprod'
```

根据需要定制的网络，通过设置每一个权重延时属性，可以指定权重延时线。

```
net.inputWeights{2,1}.delays = [0 1];
net.inputWeights{2,2}.delays = 1;
net.layerWeights{3,3}.delays = 1;
```

8. 网络函数

再次输入 net 命令，观察接下来的属性：

```
functions:
adaptFcn: (none)
initFcn: (none)
performFcn: (none)
trainFcn: (none)
```

每一个这样的属性都定义了一个基本网络操作的函数。

设置初始化函数为 initlay，这样网络根据层初始化函数来初始化自身，而层初始化函数就是已经设置为 initnw 的 Nguyen-Widrow 初始化函数。

```
net.initFcn = 'initlay';
```

这就完成了需要的初始化工作。

按照定制网络的最终需要，设置性能函数为 mse（均方误差），并且设置训练函数为 trainlm（Levenberg-Marquardt 回传）。

```
net.performFcn = 'mse';
net.trainFcn = 'trainlm';
```

这时，相应的网络函数属性变为：

```
functions:
```

```
adaptFcn: (none)
initFcn: 'initlay'
performFcn: 'mse'
trainFcn: 'trainlm'
```

9．权重及偏置值

在初始化和训练网络之前，观察一下最后一组网络属性（除了 userdata 属性之外）。

```
weight and bias values:
IW: {3x2 cell} containing 3 input weight matrices
LW: {3x3 cell} containing 3 layer weight matrices
b: {3x1 cell} containing 2 bias vectors
```

这些细胞数组在某些的位置包含了权重矩阵和偏置向量，这些位置（即连接属性）（net.inputConnect、net.layerConnect、net.biasConnenct）包含 1，以及子对象属性（net.inputWeights、net.layerWeights、net.biases）包含结构体的位置。

运行下面的代码，可以看到所有的偏置向量和权重矩阵均为 0。

```
net.IW{1,1}
net.IW{2,1}
net.IW{2,2}
net.IW{3,1}
net.LW{3,2}
net.LW{3,3}
net.b{1}
net.b{3}
```

每个输入权重 net.IW{i,j}、层权重 net.LW{i,j}，以及偏置向量 net.b{i}都有与第 i 层（net.layers{i}.size）大小一样的行数。

每个输入权重 net.IW{i,j}的列数与第 j 个输入（net.inputs{j}.size）乘以其延时数目（length(net.inputWeights{i,j}.delays)）相同。

同样地，每层的权重其列数与第 j 层的大小（net.layers{j}.size）乘以其延时数目（length(net.layerWeights{i,j}.delays)）相同。

这时，使用 net 命令，网络属性变为：

```
net=
NeuralNetworkobject:
architecture:
numInputs:2
numLayers:3
biasConnect:[1;0;1]
inputConnect:[10;11;00]
layerConnect:[000;000;111]
outputConnect:[011]
targetConnect:[001]
numOutputs:2(read-only)
numTargets: 1   (read-only)
numInputDelays: 0   (read-only)
numLayerDelays: 0   (read-only)
```

```
subobject structures:
inputs: {2x1 cell} of inputs
layers: {3x1 cell} of layers
outputs: {1x3 cell} containing 2 outputs
targets: {1x3 cell} containing 1 target
biases: {3x1 cell} containing 2 biases
inputWeights: {3x2 cell} containing 3 input weights
layerWeights: {3x3 cell} containing 3 layer weights
functions:
adaptFcn: (none)
initFcn: 'initlay'
performFcn: 'mse'
trainFcn: 'trainlm'
parameters:
adaptParam: (none)
initParam: (none)
performParam: (none)
trainParam: .epochs, .goal, .max_fail, .mem_reduc,
.min_grad, .mu, .mu_dec, .mu_inc,
.mu_max, .show, .time
weight and bias values:
IW: {3x2 cell} containing 3 input weight matrices
LW: {3x3 cell} containing 3 layer weight matrices
b: {3x1 cell} containing 2 bias vectors
other:
userdata: (user stuff)
```

13.1.3 网络行为

1．初始化

用下面的代码初始化网络。

```
net = init(net)
```

再次查看网络的偏置和权重，可以发现它们产生了变化。

```
net.IW{1,1}
net.IW{2,1}
net.IW{2,2}
net.IW{3,1}
net.LW{3,2}
net.LW{3,3}
net.b{1}
net.b{3}
```

得到下面的结果：

```
>> net.IW{1,1}
ans =
0.4226     0.3674
```

```
-0.4010    0.3909
0.5186    -0.2113
-0.0163   -0.5598
>> net.IW{2,1}
ans =
0.6428    0.5839    -0.6475    0.8338
-0.1106   0.8436    -0.1886    -0.1795
0.2309    0.4764    0.8709     0.7873
>> net.IW{2,2}
ans =
-0.8842   -0.9803   -0.6026   -0.6024   -0.1098
-0.2943   -0.7222   0.2076    -0.9695   0.8636
0.6263    -0.5945   -0.4556   0.4936    -0.0680
>> net.IW{3,1}
ans =
[]
>> net.LW{3,2}
ans =
0.3443    0.6762    -0.9607
>> net.LW{3,3}
ans =
0.3626
>> net.b{1}
ans =
-6.7502
0.9838
-0.6032
0.0803
>> net.b{3}
ans =
-0.2410
```

2．训练

对两个时间步长（如两列）定义下面的细胞数组，由两个输入向量组成（一个具有两个元素，一个具有五个元素）。

```
P = {[0; 0] [2; 0.5]; [2; -2; 1 ;0; 1] [-1; -1; 1; 0; 1]}
```

得到结果：

```
P =
[2x1 double]    [2x1 double]
[5x1 double]    [5x1 double]
```

希望网络对下面的目标序列进行响应：

```
T = {1 -1}
```

即：

```
T =
[1]    [-1]
```

在训练之前，对网络进行仿真，来观察初始的网络响应 Y 是否与目标 T 相近。

输入下面的代码：

```
Y = sim(net,P)
```

得到如下结果：

```
Y =
[3x1 double]    [3x1 double]
[     0.3994]   [     0.4319]
```

这个细胞数组 Y 的第二行是第二个网络输出的输出序列，它也是第三层的输出序列。在具体计算时，得到的第二行的值可能与这里给出的值不一样，这是因为初始的权重，以及偏置不成造成的。但是，它们一般来说都不太可能与目标 T 相同。

下面的任务就是准备训练参数了。下面的代码显示了默认的 Levenberg-Marquardt 训练参数（这是在设置 net.trainFcn 为 trainlm 时定义的）。

```
net.trainParam
```

得到如下的结果：

```
ans =
epochs: 100
goal: 0
max_fail: 5
mem_reduc: 1
min_grad: 1.0000e-010
mu: 0.0010
mu_dec: 0.1000
mu_inc: 10
mu_max: 1.0000e+010
show: 25
time: Inf
```

用下面的代码，改变性能目标为：1e-10。

```
net.trainParam.goal = 1e-10;
```

接下来，使用下面的调用训练网络：

```
net = train(net,P,T);
```

在训练过程中显示如下结果：

```
TRAINLM, Epoch 0/100, MSE 1.20559/1e-010, Gradient 3.47128/1e-010
TRAINLM, Epoch 3/100, MSE 1.38138e-011/1e-010, Gradient 1.5634e-005/1e-010
TRAINLM, Performance goal met.
```

训练结束时，绘制出训练图，如图 13-2 所示。

从图 13-2 中可以看出，经过三步训练，就达到了设定的性能目标，这与训练过程中显示的步数也是吻合的。

如果改变网络的性能目标，将性能要求提高，则需要更多的训练步数和训练时间，在实际中需要根据要求合理地选择性能目标，以达到性能与时间的综合优化。

训练完网络后，可以对这个训练过的网络进行仿真，检验一下训练的网络是否学会了正确地响应。

输入如下的代码：

```
Y = sim(net,P)
```

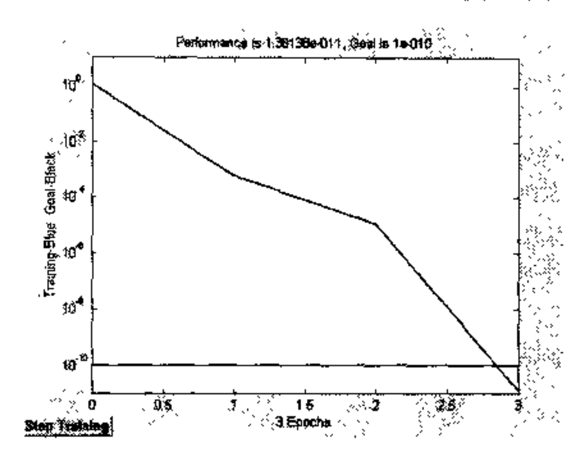


图 13-2 训练过程图

可以得到如下的结果：

Y =

[3x1 double] [3x1 double]

[1.0000] [-1.0000]

显然，第二个网络输出（即细胞数组 Y 的第二行）也作为第三层的输出，与目标序列 T 很好地吻合。这说明训练后的网络能够得到很好的仿真结果。

13.2 附加的工具箱函数

前面的章节介绍了很多工具箱函数，这些都是在使用工具箱网络时使用的。但是，还有一些工具箱函数，它们不被工具箱网络使用，而可以在创建自己定制的网络时使用，本节主要介绍在 MATLAB 中包含的这样一些函数。

本节主要分三个部分介绍这些函数：

- 初始化函数
- 传递函数
- 学习函数

13.2.1 初始化函数

1．randnc

功能 这个权重初始化函数产生随机的权重矩阵，其列的长度被规范化为 1。

格式 W = randnc(S,PR)

W = randnc(S,R)

说明 randnc 是一个权重初始化函数。其输入参数为：

- s 行（神经元）的数目
- PR R×2 的矩阵，其值的范围为[Pmin Pmax]。

该函数执行之后返回一个 S×R 的随机矩阵，其列被规范化为 1。

参考 randnr。

2．randnr

功能 这个权重初始化函数产生随机的权重矩阵，其行的长度被规范化为 1。

格式 W = randnr(S,PR)

W = randnr(S,R)

说明 randnc 是一个权重初始化函数。其输入参数为：

- s 行（神经元）的数目
- PR R×2 的矩阵，其值的范围为[Pmin Pmax]。

该函数执行之后返回一个 S×R 的随机矩阵，其行被规范化为 1。

参考 randnc。

13.2.2 传递函数

1．satlin

功能 这个传递函数的功能与 satlins 类似，但是它有一个从 0～1 的线性区域（而不是从－1～1），其最小值和最大值是 0 和 1（而不是－1 和 1）。

格式 A = satlin(N)

info = satlin(code)

说明 satlin 是一个传递数，它有一个输入参数：

N S×Q 的网络输入（列）向量

该函数执行之后将返回值截到区间[-1，1]之间。

下面的代码绘制出了使用 satlin 传递函数得到的图。

```
n = -5:0.1:5;
a = satlin(n);
plot(n,a)
```

图 13-3 是得到的传递函数图。

网络使用 改变一个网络，使得它的一层使用 satlin，设置 net.layers{i}.transferFcn 为 'satlin'。调用 sim 函数来仿真使用了 satlin 的网络。

算法 satlin(n) = 0，如果 $n <= 0$

n，如果 $0 <= n <= 1$

1，如果 $1 <= n$

参考 sim、poslin、satlins、purelin。

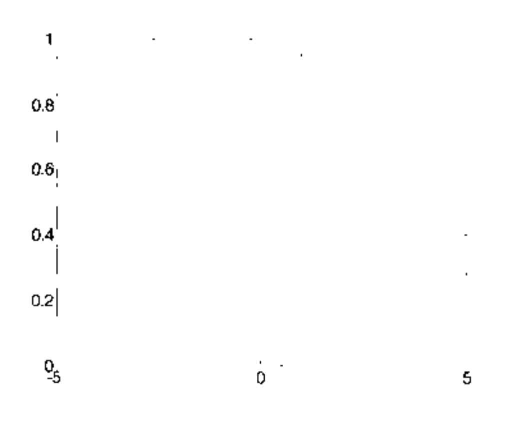


图 13-3 satlin 传递函数

2．softmax

功能 这个传递函数是“硬”竞争传递函数 compet 的一个“软”的版本。具有最大网络输入的神经元得到一个与 1 很接近的输出，而其他的神经元得到与 0 接近的输出。

格式 A = softmax(N)

info = softmax(code)

说明 softmax 是一个传递函数函数。它有一个输入参数：

N S×Q 的网络输入（列）向量

该函数执行之后返回一个输出向量，其元素在 0 和 1 之间。

下面的代码定义了一个网络输入向量 *N*，并且计算输出，绘制出它们的条状图像。

```
n = [0; 1; -0.5; 0.5];
a = softmax(n);
subplot(2,1,1), bar(n), ylabel('n')
subplot(2,1,2), bar(a), ylabel('a')
```

得到图像如图 13-4 所示。

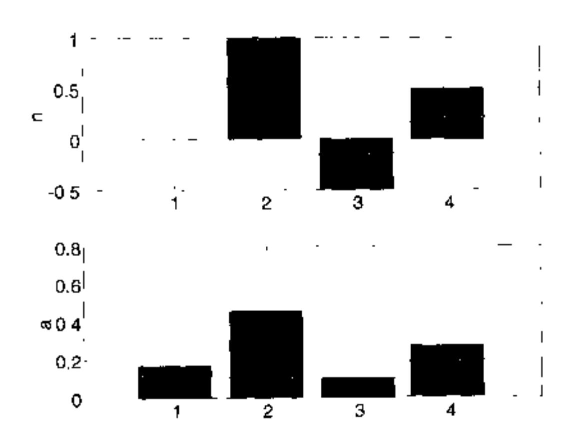


图 13-4 softmax 传递函数变换得到的图像

网络使用 改变一个网络，使得它的某一层使用 softmax 传递函数，设置 net.layers{i,j}.transferFcn 为'softmax'。然后调用 sim 函数来仿真使用了 softmax 传递函数的网络。

参考 sim、compet

3．tribas

功能 这是一个三角基传递函数，类似于径向基传递函数 radbas，但是，它的形状简单。

格式 A = tribas(N)

info = tribas(code)

说明 tribas 是一个传递函数。它只有一个输入参数：

N S×Q 的网络输入（列）向量

该函数执行之后返回 N 通过一个径向基函数后得到的每个元素。

下面的代码绘制出了一个使用 tribas 传递函数得到的图像。

```
n = -5:0.1:5;
a = tribas(n);
plot(n,a)
```

得到的图像如图 13-5 所示。

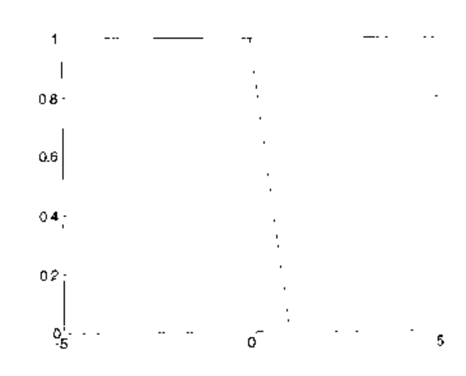


图 13-5 使用 tribas 传递函数得到的图像

网络使用 改变网络，使得它的某层使用 tribas 传递函数，设置 net.layers{i}.transferFcn 为'tribas'。调用 sim 对改变的网络进行仿真。

算法 tribas(N)按照下面算法计算输出：

tribas(n) = 1 - abs(n)，如果 $-1 <= n <= 1$

= 0，其他

参考 sim、radbas。

13.2.3 学习函数

1．learnh

功能 这个 Hebb 权重学习函数按照乘积、权重输入，以及神经元输出的比例来增加权重。它允许神经元学习输入与输出之间的相关性。

格式 [dW,LS] = learnh(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)

```
info = learnh(code)
```

说明 learnh(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)函数具有多个输入参数：

- W S×R 权重矩阵（或者 S×1 偏置向量）
- P R×Q 输入向量（或者 ones(1,Q)）
- Z S×Q 加权输入向量
- N S×Q 网络输入向量
- A S×Q 输出向量
- T S×Q 层目标向量
- E S×Q 层误差向量
- gW S×R 性能相关的梯度
- gA S×Q 性能相关的输出梯度
- D S×S 神经元距离
- LP 学习参数，空，LP = []
- LS 学习状态，初始时应该为[]

其返回值为：

- dW S×R 权重（或者偏置）改变矩阵
- LS 新的学习状态

学习是按照 learnh 的学习参数来进行的，这里列出其默认值：

LP.lr 0.01 学习速率

learnh(code)对于每个 code 字符串返回有用的信息：

- 'pnames' 返回学习参数的名称
- 'pdefaults' 返回默认的学习参数
- 'needg' 如果这个函数使用 gW 或者 gA，则返回 1

下面的代码为一个层定义了一个随机输入 P 和输出 A，它具有一个两元素的输入和三个神经元，同时，还定义了学习速率 lr。

```
p = rand(2,1);
a = rand(3,1);
lp.lr = 0.5;
```

由于 learnh 只需要这些值就可以计算一个权重变化（可以参照下面给出的算法），因此可以用以下的代码来完成这个任务。

```
dW = learnh([],p,[],[],a,[],[],[],[],[],lp,[])
```

得到权重改变矩阵如下：

```
dW =
0.2883    0.0701
0.2309    0.0562
0.4234    0.1030
```

网络使用 为了准备将某个定制网络第 i 层的权重和偏置来用 learnh 进行学习，需要以下步骤：

- 设置 net.trainFcn 为'trainr'（net.trainParam 将自动变成 trainr 的默认参数）。
- 设置 net.adaptFcn 为'trains'（net.adaptParam 将自动变成 trainr 的默认参数）。

- 设置每个 net.inputWeights{i,j}.learnFcn 为'learnh'，设置每个 net.layerWeights{i,j}.learnFcn 为'learnh'（每个权重学习参数属性会自动地设置为 learnh 的默认参数）。

为了训练网络（或者使其能够自调整），需要进行以下操作：

- 设置 net.trainParam（net.adaptParam）属性为需要的值。
- 调用 train(adapt)。

算法 learnh 函数按照 Hebb 学习规则为神经元的输入 P、输出 A，以及学习速率 lr 计算权重改变 dW。

```
dw = lr*a*p'
```

参考 learnhd、adapt、train。

2．learnhd

功能 这个退化的 Hebb 学习函数类似于 Hebb 函数，但是增加了一项，它随着时间步长指数的对权重递减。这个权重退化允许神经元忘掉那些没有固定强化的相关性，这也解决了 Hebb 函数具有无界增长的权重的问题。

格式 [dW,LS] = learnhd(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)

info = learnhd(code)

说明 learnhd 是 Hebb 权重学习函数。

learnhd(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)函数具有多个输入参数：

- W　S×R 权重矩阵（或者 S×1 偏置向量）
- P　R×Q 输入向量（或者 ones(1,Q)）
- Z　S×Q 加权输入向量
- N　S×Q 网络输入向量
- A　S×Q 输出向量
- T　S×Q 层目标向量
- E　S×Q 层误差向量
- gW　S×R 性能相关的梯度
- gA　S×Q 性能相关的输出梯度
- D　S×S 神经元距离
- LP　学习参数，空，LP = []
- LS　学习状态，初始时应该为[]

它的返回值为：

- dW　S×R 权重（或者偏置）改变矩阵
- LS　新的学习状态

学习是按照 learnhd 的学习参数来进行的，这里列出其默认值：

- LP.dr　0.01　退化速率
- LP.lr　0.1　学习速率

learnhd(code)对于每个 code 字符串返回有用的信息。

- 'pnames'　返回学习参数的名称

- 'pdefaults' 返回默认的学习参数
- 'needg' 如果这个函数使用 gW 或者 gA，则返回 1

下面的代码为一个层定义了一个随机输入 P 和输出 A，它具有一个两元素的输入和三个神经元，同时，还定义了退化速率 dr 和学习速率 lr。

```
p = rand(2,1);
a = rand(3,1);
w = rand(3,2);
lp.dr = 0.05;
lp.lr = 0.5;
```

由于 learnhd 只需要这些值就可以计算一个权重变化（可以参照下面给出的算法），因此可以用以下的代码来完成这个任务。

```
dW = learnhd(w,p,[],[],a,[],[],[],[],[],lp,[])
```

得到权重改变矩阵如下：

```
dW =
0.0378    0.0061
-0.0420   -0.0402
0.2022    0.0754
```

网络使用 为了准备将某个定制网络第 *i* 层的权重和偏置来用 learnhd 进行学习，需要以下步骤：

- 设置 net.trainFcn 为'trainr'（net.trainParam 将自动变成 trainr 的默认参数）。
- 设置 net.adaptFcn 为'trains'（net.adaptParam 将自动变成 trainr 的默认参数）。
- 设置每个 net.inputWeights{i,j}.learnFcn 为'learnhd'，设置每个 net.layerWeights{i,j}.learnFcn 为'learnhd'（每个权重学习参数属性会自动地设置为 learnhd 的默认参数）。

为了训练网络（或者使其能够自调整），需要进行以下操作：

- 设置 net.trainParam（net.adaptParam）属性为需要的值。
- 调用 train(adapt)。

算法 learnhd 函数按照 Hebb 退化学习规则为神经元的输入 P、输出 A，以及学习速率 lr 计算权重改变 dW。

```
dw =   lr*a*p' - dr*w
```

参考 learnh、adapt、train。

3．learnis

功能 这个内星权重学习函数按照神经元输出的步长比例，将神经元的权重向量朝着神经元输入向量的方向移动。这个函数允许神经元学习输入向量与其输出之间的相关性。

格式 [dW,LS] = learnis(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)

info = learnis(code)

说明 learnis 是内星权重学习函数。

learnis(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)函数具有多个输入参数：

- W　S×R 权重矩阵（或者 S×1 偏置向量）
- P　R×Q 输入向量（或者 ones(1,Q)）

- Z　S×Q 加权输入向量
- N　S×Q 网络输入向量
- A　S×Q 输出向量
- T　S×Q 层目标向量
- E　S×Q 层误差向量
- gW　S×R 性能相关的梯度
- gA　S×Q 性能相关的输出梯度
- D　S×S 神经元距离
- LP　学习参数，空，LP = []
- LS　学习状态，初始时应该为[]

其返回值为：

- dW　S×R 权重（或者偏置）改变矩阵
- LS　新的学习状态

学习是按照 learnis 的学习参数来进行的，这里列出其默认值：

LP.lr　0.01　学习速率

learnis(code)对于每个 code 字符串返回有用的信息：

- 'pnames'　返回学习参数的名称
- 'pdefaults'　返回默认的学习参数
- 'needg'　如果这个函数使用 gW 或者 gA，则返回 1

下面的代码为一个层定义了一个随机输入 P 和输出 A，它具有一个两元素的输入和三个神经元，同时还定义了学习速率 lr。

```
p = rand(2,1);
a = rand(3,1);
w = rand(3,2);
lp.lr = 0.5;
```

由于 learnis 只需要这些值就可以计算一个权重变化（可以参照下面给出的算法），因此可以用以下的代码来完成这个任务。

```
dW = learnis(w,p,[],[],a,[],[],[],[],[],lp,[])
```

得到权重改变矩阵如下：

```
dW =
0.1195    0.1596
0.0546    0.3191
0.0268    0.0208
```

网络使用　为了准备将某个定制网络第 i 层的权重和偏置来用 learnis 进行学习，需要以下步骤：

- 设置 net.trainFcn 为'trainr'（net.trainParam 将自动变成 trainr 的默认参数）。
- 设置 net.adaptFcn 为'trains'（net.adaptParam 将自动变成 trainr 的默认参数）。
- 设置每个 net.inputWeights{i,j}.learnFcn 为 'learnis'，设置每个 net.layerWeights{i,j}.learnFcn 为'learnis'（每个权重学习参数属性会自动地设置为 learnis 的默认参数）。

为了训练网络（或者使其能够自调整），需要进行以下操作：

- 设置 net.trainParam（net.adaptParam）属性为需要的值。
- 调用 train(adapt)。

算法 learnis 函数按照内星学习规则为神经元的输入 P、输出 A，以及学习速率 LR 计算权重改变 dW。

dw = lr*a*(p'-w)

参考 learnk、learnos、adapt、train。

4．learnos

功能 这个外星权重学习函数按照与内星学习规则相对的方式运行。外星学习规则根据神经元输入值的步长比例，将神经元的权重向量从神经元输入向量的方向朝着神经元某层输出向量的方向移动。这个函数在仿真时允许输入学习调用向量。

格式 [dW,LS] = learnos(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)

info = learnos(code)

说明 learnos 是外星权重学习函数。

learnos(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)函数具有多个输入参数：

- W S×R 权重矩阵（或者 S×1 偏置向量）
- P R×Q 输入向量（或者 ones(1,Q)）
- Z S×Q 加权输入向量
- N S×Q 网络输入向量
- A S×Q 输出向量
- T S×Q 层目标向量
- E S×Q 层误差向量
- gW S×R 性能相关的梯度
- gA S×Q 性能相关的输出梯度
- D S×S 神经元距离
- LP 学习参数，空，LP = []
- LS 学习状态，初始时应该为[]

它的返回值为：

- dW S×R 权重（或者偏置）改变矩阵
- LS 新的学习状态

学习是按照 learnos 的学习参数来进行的，这里列出其默认值：

LP.lr 0.01 学习速率

learnos(code)对于每个 code 字符串返回有用的信息：

- 'pnames' 返回学习参数的名称
- 'pdefaults' 返回默认的学习参数
- 'needg' 如果这个函数使用 gW 或者 gA，则返回 1

下面的代码为一个层定义了一个随机输入 P 和输出 A，它具有一个两元素的输入和三个神经元，同时，还定义了学习速率 lr。

```
p = rand(2,1);
a = rand(3,1);
w = rand(3,2);
lp.lr = 0.5;
```

由于 learnos 只需要这些值就可以计算一个权重变化（可以参照下面给出的算法），因此可以用以下的代码来完成这个任务。

```
dW = learnos(w,p,[],[],a,[],[],[],[],[],lp,[])
```

得到权重改变矩阵如下：

```
dW =
0.0465    0.0432
-0.0823   -0.1375
0.0181    0.1266
```

网络使用 为了准备将某个定制网络第 i 层的权重和偏置采用 learnos 进行学习，需要以下步骤：

- 设置 net.trainFcn 为'trainr'（net.trainParam 将自动变成 trainr 的默认参数）。
- 设置 net.adaptFcn 为'trains'（net.adaptParam 将自动变成 trainr 的默认参数）。
- 设置每个 net.inputWeights{i,j}.learnFcn 为'learnos'，设置每个 net.layerWeights{i,j}.learnFcn 为'learnos'（每个权重学习参数属性会自动地设置为 learnos 的默认参数）。

为了训练网络（或者使其能够自调整），需要进行以下操作：

- 设置 net.trainParam（net.adaptParam）属性为需要的值。
- 调用 train(adapt)。

算法 learnos 函数按照外星学习规则为神经元的输入 P、输出 A，以及学习速率 LR 计算权重改变 dW。

```
dw = lr*(a-w)*p'
```

参考 learnk、learnis、adapt、train。

13.3 定制函数

工具箱允许生成并使用很多种类的函数，这使得在使用初始化、仿真，以及训练的算法时有很多方法，并且允许对网络进行自调整。

在下面的内容里，将分四个部分介绍怎样生成自己定制的这些函数：

- 仿真函数
  - 传递函数
  - 网络输入函数
  - 权重函数
- 初始化函数
  - 网络初始化函数
  - 层初始化函数
  - 权重及偏置初始化函数

- 学习函数
    - 网络训练函数
    - 网络自调整函数
    - 网络性能函数
    - 权重及偏置学习函数
- 自组织映射函数
    - 拓扑函数
    - 距离函数

13.3.1 仿真函数

可以生成三种仿真函数：传递函数、网络输入函数，以及权重函数。同样，也可以提供相关的分派函数，使得回传学习可以实现。

1．传递函数

传递函数根据给定的网络输入向量（或者矩阵）***N***，计算某层的输出向量（或者矩阵）***A***。输出和网络输入之间关系的惟一限制就是输出必须与输入具有相同的维数。

传递函数一旦得到了定义，就可以指定用到网络的任何一层。例如，下面的代码将传递函数 yourtf 指定到某个网络的第二层。

```
net.layers{2}.transferFcn = yourtf';
```

于是，任何时候仿真网络，这个传递函数都被使用。

```
[Y,Pf,Af] = sim(net,P,Pi,Ai)
```

作为一个合法的传递函数，函数必须计算从网络输入 N 得到的输出 A，如下所示：

```
A = yourtf(N)
```

这里：

- ***N*** 是一个 S×Q 的矩阵，它具有 *Q* 个网络输入（列）向量。
- ***A*** 是一个 S×Q 的矩阵，它具有 *Q* 个输出（列）向量。

传递函数还必须提供关于自身的信息，可使用下面的调用格式：

```
info = yourtf(code)
```

对于每一个下列的字符串代码，返回相应的信息：

- 'version' 返回神经网络工具箱版本
- 'deriv' 返回相关的分派函数名字
- 'output' 返回输出范围
- 'active' 返回起作用的输入范围

在工具箱中包含了一个名字为 mytf 的定制传递函数的例子。输入下面的代码，观察怎样使用。

首先输入：

```
help mytf
```

可以得到以下的帮助信息：

```
MYTF Example custom transfer function.
Use this function as a template to write your own function.
```

```
Calculation Syntax
A = mytf(N)
N - S×Q matrix of Q net input (column) vectors.
A - S×Q matrix of Q output (column) vectors.
Information Syntax
info = mytf(code) returns useful information for each CODE string:
'version' - Returns the Neural Network Toolbox version (3.0).
'deriv'   - Returns the name of the associated derivative function.
'output'  - Returns the output range.
'active'  - Returns the active input range.
Example
n = -5:.1:5;
a = mytf(n);
plot(n,a)
$Revision: 1.1 $
```

输入下面的代码：

```
n = -5:.1:5;
a = mytf(n);
plot(n,a)
mytf('deriv')
```

可以得到输出：

```
ans =
mydtf
```

并得到如图 13-6 所示的图像。

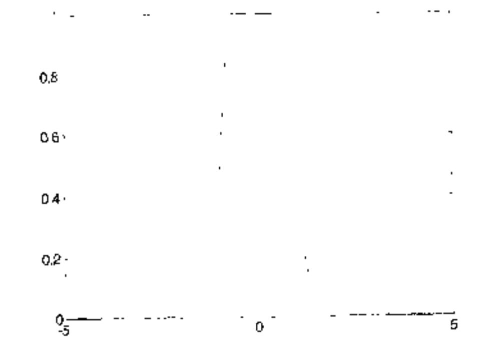


图 13-6　函数 mytf 的曲线

输入下列代码，可以看到 mytf 函数的具体实现。

```
type mytf
```

其具体实现为：

```
function a = mytf(n)
%MYTF Example custom transfer function.
%
%   Use this function as a template to write your own function.
%
```

```
%    Calculation Syntax
%
%      A = mytf(N)
%        N - S×Q matrix of Q net input (column) vectors.
%        A - S×Q matrix of Q output (column) vectors.
%
%    Information Syntax
%
%      info = mytf(code) returns useful information for each CODE string:
%        'version' - Returns the Neural Network Toolbox version (3.0).
%        'deriv'   - Returns the name of the associated derivative function.
%        'output'  - Returns the output range.
%        'active'  - Returns the active input range.
%
%    Example
%
%      n = -5:.1:5;
%      a = mytf(n);
%      plot(n,a)
% $Revision: 1.1 $
if nargin < 1, error('Not enough arguments.'); end
if isstr(n)
switch (n)
case 'version'
a = 3.0;          % <-- Must be 3.0.
case 'deriv'
a = 'mydtf';      % <-- Replace with the name of your
%          associated function or ''
case 'output'
a = [-1 1];       % <-- Replace with the minimum and maximum
%          output values of your transfer function
case 'active'
a = [-2 2];       % <-- Replace with the range of inputs where
%          the outputs are most sensative to changes.
otherwise, error('Unrecognized code.')
end
else
a = 1./(n.^8+1);    % <-- Replace with your calculation
end
```

可以使用 mytf 函数作为一个模板来生成自己的传递函数。

下面来讲传递分派函数。

如果需要在定制的传递函数上使用回传，则需要为其生成一个定制的分派函数。这个函数需要根据其网络输入计算此层输出的分派。

```
dA_dN = yourdtf(N,A)
```

这里：

- N 是一个 S×Q 的矩阵，它具有 Q 个网络输入（列）向量。
- A 是一个 S×Q 的矩阵，它具有 Q 个输出（列）向量。
- dA_dN 是 S×Q 的分派 dA/dN。

这只对于那些输出元素相互独立的传递函数起作用。换句话说，这里的每个 A(i)只是 N(i)的一个函数。否则，在多向量的情况下，需要一个三维数组（而不是前面定义的矩阵）来存储这些分派。这样的三维分派现在还不支持。

在工具箱中包含了一个名字为 mydtf 的例子。输入下面的代码，观察怎样使用。

首先输入：

```
help mydtf
```

得到下面的帮助信息：

```
MYDTF Example custom transfer derivative function of MYTF.
Use this function as a template to write your own function.
Syntax
dA_dN = mydtf(N,A)
N - SxQ matrix of Q net input (column) vectors.
A - SxQ matrix of Q output (column) vectors.
dA_dN - SxQ derivative dA/dN.
Example
n = -5:.1:5;
a = mytf(n);
da_dn = mydtf(n,a);
subplot(2,1,1), plot(n,a)
subplot(2,1,2), plot(n,da_dn)
$Revision: 1.1 $
```

接下来输入下面的代码：

```
da_dn = mydtf(n,a);
subplot(2,1,1), plot(n,a)
subplot(2,1,2), plot(n,da_dn)
```

结果如图 13-7 所示。

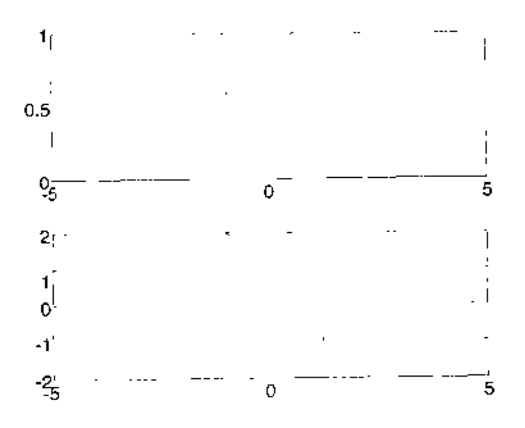


图 13-7　函数 mydtf 的曲线

使用下面的命令，可以显示出函数 mydtf 的实现。

```
type mydtf
```

得到函数 mydtf 的实现如下：

```
function d = mydtf(n,a)
%MYDTF Example custom transfer derivative function of MYTF.
%
%   Use this function as a template to write your own function.
%
%   Syntax
%
%     dA_dN = mydtf(N,A)
%       N - S×Q matrix of Q net input (column) vectors.
%       A - S×Q matrix of Q output (column) vectors.
%       dA_dN - S×Q derivative dA/dN.
%
%   Example
%
%     n = -5:.1:5;
%     a = mytf(n);
%     da_dn = mydtf(n,a);
%     subplot(2,1,1), plot(n,a)
%     subplot(2,1,2), plot(n,da_dn)
% $Revision: 1.1 $

% **  Replace the following calculation with your
% **  derivative calculation.

d = -8*n.^7.*a.^2;

% **  Note that you have both the transfer functions input N and
% **  output A available, which can often allow a more efficient
% **  calculation of the derivative than with just N.
```

可以使用 mydtf 函数作为一个模板来生成自己的传递分派函数。

2．网络输入函数

网络输入函数根据给定的加权输入向量（矩阵）***Zi*** 计算某层的网络输入向量（或者矩阵）***N***。网络输入和加权输入之间的惟一的限制就是网络输入必须与加权输入有相同的维数，并且函数不能对加权输入的顺序敏感。

函数定义以后，就可以将这个网络输入函数指定到某个网络的任意一层上。例如，下面的代码指定输入函数 yournif 到一个网络的第二层上。

```
net.layers{2}.netInputFcn = 'yournif';
```

这样，任何时候对这个网络进行仿真时，网络输入函数都会被使用。

```
[Y,Pf,Af] = sim(net,P,Pi,Ai)
```

作为一个合法的网络输入函数，函数必须计算从网络输入 N 得到的输出 A，如下所示：

```
N = yournif(Z1,Z2,...)
```

这里：

- Zi 是个加权输入（列）向量的第 *i* 个 S×Q 的矩阵。
- N 是 Q 个网络输入（列）向量的一个 S×Q 的矩阵。

网络输入函数也必须使用下面的调用格式提供关于自身的信息。

```
info = yournif(code)
```

对于每一个下列的字符串代码，返回相应的信息：

- 'version'　返回神经网络工具箱版本。
- 'deriv'　返回相关的分派函数名字。

在工具箱中包含了一个名字为 mynif 的定制网络输入函数的例子。输入下面的代码，观察怎样使用。

首先输入：

```
help mynif
```

可以得到以下的帮助信息：

```
MYNIF Example custom net input function.
Use this function as a template to write your own function.
Calculation Syntax
N = mynif(Z1,Z2,...)
Zi - SxQ matrix of Q weighted (column) vectors.
N - SxQ matrix of Q net input (column) vectors.
Information Syntax
info = mynif(code) returns useful information for each CODE string:
'version' - Returns the Neural Network Toolbox version (3.0).
'deriv'    - Returns the name of the associated derivative function.
Example
z1 = rand(4,5);
z2 = rand(4,5);
z3 = rand(4,5);
n = mynif(z1,z2,z3)
$Revision: 1.1 $
```

接下来输入下面的代码：

```
z1 = rand(4,5);
z2 = rand(4,5);
z3 = rand(4,5);
n = mynif(z1,z2,z3)
mynif('deriv')
```

得到下面的结果：

```
n =
0.1176    0.1159    0.1804    0.2213    0.1437
0.0721    0.0789    0.0554    0.1972    0.1200
0.0111    0.2290    0.2322    0.1760    0.1146
0.1781    0.1252    0.2005    0.1074    0.1678
ans =
deriv
```

使用下面的命令，可以显示出函数 mynif 的实现。

```
type mynif
```

得到函数 mynif 的实现如下：

```
function n=mynif(varargin)
%MYNIF Example custom net input function.
%
%    Use this function as a template to write your own function.
%
%    Calculation Syntax
%
%       N = mynif(Z1,Z2,...)
%          Zi - S×Q matrix of Q weighted (column) vectors.
%          N - S×Q matrix of Q net input (column) vectors.
%
%    Information Syntax
%
%       info = mynif(code) returns useful information for each CODE string:
%          'version' - Returns the Neural Network Toolbox version (3.0).
%          'deriv'     - Returns the name of the associated derivative function.
%
%    Example
%
%       z1 = rand(4,5);
%       z2 = rand(4,5);
%       z3 = rand(4,5);
%       n = mynif(z1,z2,z3)
% $Revision: 1.1 $
if nargin < 1, error('Not enough arguments.'); end
n = varargin{1};
if isstr(n)
switch n
case 'version'
a = 3.0;            % <-- Must be 3.0.
case 'deriv'
a = 'mydnif';   % <-- Replace with the name of your
%        associated derivative function or ''
otherwise
error('Unrecognized code.')
end
else
% **   Replace the following calculation with your own.   The only
% **   constraint is that the function must not be sensative
% **   to the order of its input arguments.
% **   In other words, MYNIF(Z1,Z2,Z3) must return the same
% **   values as MYNIF(Z2,Z3,Z1), MYNIF(Z1,Z3,Z2), etc.
n = 1./n;
```

```
for i=2:length(varargin)
n = n + 1./varargin{i};
end
n = 1./n;
end
```

可以使用 mynif 函数作为一个模板来生成自己的网络输入函数。

现在来讲网络输入分派函数。

如果需要在定制的网络输入函数上使用回传，则需要为其生成一个定制的分派函数。这个函数需要根据其加权输入计算此层的网络输入分派。

```
dN_dZ = dtansig(Z,N)
```

这里：

- Z 是一个 S×Q 的矩阵，它具有 Q 个网络输入（列）向量。
- A 是一个 S×Q 的矩阵，它具有 Q 个输出（列）向量。
- dN_dZ 是 S×Q 的分派 dN/dZ。

在工具箱中包含了一个名字为 mydnif 的例子。输入下面的代码，观察怎样使用。

首先输入：

```
help mydnif
```

得到下面的帮助信息：

```
MYDNIF Example custom net input derivative function of MYNIF.
Use this function as a template to write your own function.
Syntax
dN_dZ = dtansig(Z,N)
Z - SxQ matrix of Q weighted input (column) vectors.
N - SxQ matrix of Q net input (column) vectors.
dN_dZ - SxQ derivative dN/dZ.
Example
z1 = rand(4,5);
z2 = rand(4,5);
z3 = rand(4,5);
n = mynif(z1,z2,z3)
dn_dz1 = mydnif(z1,n)
dn_dz2 = mydnif(z2,n)
dn_dz3 = mydnif(z3,n)
$Revision: 1.2 $
```

接下来输入下面的代码：

```
z1 = rand(4,5);
z2 = rand(4,5);
z3 = rand(4,5);
n = mynif(z1,z2,z3)
dn_dz1 = mydnif(z1,n);
dn_dz2 = mydnif(z2,n);
dn_dz3 = mydnif(z3,n);
```

得到下面的结果。

```
n =
```

```
0.1884    0.1752    0.0318    0.0140    0.0431
0.0857    0.0123    0.0310    0.0158    0.1860
0.0371    0.1472    0.0443    0.0136    0.2253
0.0243    0.2459    0.1814    0.0954    0.1569
dn_dz1 =
0.0146    0.0142    0.0000    0.0000    0.0017
0.0003    0.0000    0.0003    0.0002    0.0339
0.0002    0.0070    0.0000    0.0000    0.0316
0.0004    0.0382    0.0057    0.0054    0.0047
dn_dz2 =
0.0088    0.0283    0.0001    0.0000    0.0001
0.0003    0.0001    0.0002    0.0002    0.0127
0.0006    0.0037    0.0017    0.0001    0.0201
0.0001    0.0335    0.0154    0.0002    0.0034
dn_dz3 =
0.0117    0.0030    0.0000    0.0000    0.0000
0.0015    0.0000    0.0000    0.0000    0.0047
0.0000    0.0032    0.0007    0.0000    0.0202
0.0000    0.0282    0.0122    0.0031    0.0127
```

使用下面的命令，可以显示出函数 mydtf 的实现。

```
type mydnif
```

得到函数 mydnif 的实现如下：

```
function d = mydnif(z,n)
%MYDNIF Example custom net input derivative function of MYNIF.
%
%   Use this function as a template to write your own function.
%
%   Syntax
%
%     dN_dZ = dtansig(Z,N)
%       Z - S×Q matrix of Q weighted input (column) vectors.
%       N - S×Q matrix of Q net input (column) vectors.
%       dN_dZ - S×Q derivative dN/dZ.
%
%   Example
%
%     z1 = rand(4,5);
%     z2 = rand(4,5);
%     z3 = rand(4,5);
%     n = mynif(z1,z2,z3)
%     dn_dz1 = mydnif(z1,n)
%     dn_dz2 = mydnif(z2,n)
%     dn_dz3 = mydnif(z3,n)
% $Revision: 1.2 $
% ** Replace the following calculation with your
% **   derivative calculation.
```

```
d = n.^2 .* z.^2;
% **   Note that you have both the net input Z in question
% **   and output N available to calculate the derivative.
```

可以使用 mydnif 函数作为一个模板来生成自己的网络输入分派函数。

3. 权重函数

权重函数根据给定的输入向量（或矩阵）***P***，以及一个权重矩阵 ***W***，计算一个加权了的输入向量（或者矩阵）**Z**。

函数定义以后，就可以将这个权重函数指定到某个网络的任意输入权重或者层权重上。例如，下面的代码指定权重函数 yourwf 到一个网络上从第一个输入到第二层的权重上。

```
net.inputWeights{2,1}.weightFcn = 'yourwf';
```

这样，任何时候对这个网络进行仿真时，权重函数都会被使用。

```
[Y,Pf,Af] = sim(net,P,Pi,Ai)
```

作为一个合法的权重函数，函数必须计算从输入 ***P*** 和一个权重矩阵 ***W*** 得到的权重输入 **Z**，如下所示：

```
Z = yourwf(W, P)
```

这里：

- W 是个 S×R 的权重矩阵。
- P 是一个 S×Q 的矩阵，它具有 Q 个输入（列）向量。
- Z 是一个 S×Q 的矩阵，它具有 Q 个加权了的输入（列）向量。

网络输入函数必须也使用下面的调用格式提供关于自身的信息。

```
info = yourwf(code)
```

对于每一个下列的字符串代码，返回相应的信息：

- 'version'　返回神经网络工具箱版本。
- 'deriv'　返回相关的分派函数名字。

在工具箱中包含了一个名字为 mywf 的定制权重函数的例子。输入下面的代码，观察怎样使用。

首先输入：

```
help mywf
```

可以得到以下的帮助信息：

```
MYWF Example custom weight function.
Use this function as a template to write your own function.
Calculation Syntax
Z = mywf(W,P)
W - S×R weight matrix.
P - R×Q matrix of Q input (column) vectors.
Z - S×Q matrix of Q weighted input (column) vectors.
Information Syntax
info = mytf(code) returns useful information for each CODE string:
'version' - Returns the Neural Network Toolbox version (3.0).
'deriv'    - Returns the name of the associated derivative function.
Example
```

```
w = rand(1,5);
p = rand(5,1);
z = mywf(w,p)
$Revision: 1.1 $
```

接下来输入下面的代码：

```
w = rand(1,5)
p = rand(5,1)
z = mywf(w,p)
mywf('deriv')
```

得到下面的结果：

```
w =
0.1911    0.4225    0.8560    0.4902    0.8159
p =
0.4608
0.4574
0.4507
0.4122
0.9016
z =
1.0494
```

使用下面的命令，可以显示出函数 mywf 的实现。

```
type mywf
```

得到函数 mywf 的实现如下：

```
function z=mywf(w,p)
%MYWF Example custom weight function.
%
%   Use this function as a template to write your own function.
%
%   Calculation Syntax
%
%     Z = mywf(W,P)
%       W - SxR weight matrix.
%       P - RxQ matrix of Q input (column) vectors.
%       Z - SxQ matrix of Q weighted input (column) vectors.
%
%   Information Syntax
%
%     info = mytf(code) returns useful information for each CODE string:
%       'version' - Returns the Neural Network Toolbox version (3.0).
%       'deriv'   - Returns the name of the associated derivative function.
%
%   Example
%
%     w = rand(1,5);
%     p = rand(5,1);
%     z = mywf(w,p)
```

```
% $Revision: 1.1 $
if nargin < 1, error('Not enough arguments.'); end
if isstr(w)
switch (w)
case 'version'
a = 3.0;          % <-- Must be 3.0.
case 'deriv'
a = 'mydtf';    % <-- Replace with the name of your
%        associated function or ''
otherwise
error('Unrecognized code.')
end
else
% **   Replace the following calculation with your
% **   weighting calculation.   The only constraint, if you
% **   want to define a derivative function, is that Z must
% **   be a sum of i terms, where each ith term is only a
% **   a function of w(i) and p(i).
z = w*(p.^2);
end
```

可以使用 mywf 函数作为一个模板来生成自己的权重函数。

现在来介绍一下权重分派函数。

如果需要在定制的权重函数上使用回传，则需要为其生成一个定制的分派函数。这个函数需要根据输入和权重计算此层的加权输入分派。

```
dZ_dP = mydwf('p',W,P,Z)
dZ_dW = mydwf('w',W,P,Z)
```

这里：

- W 是一个 S×R 的权重矩阵。
- P 是一个 R×Q 的矩阵，其具有 Q 个输入（列）向量。
- Z 是一个 S×Q 的矩阵，其具有 Q 个加权输入（列）向量。
- dZ_dP 是 S×R 的分派 dZ/dP。
- dZ_dW 是 R×Q 的分派 dZ/dW。

这只对于那些输出包含了 i 项之和，且每个第 i 项仅仅是 W(i)和 P(i)的函数的加权函数起作用。否则，在多向量的情况下，需要一个三维数组（而不是前面定义的矩阵）来存储这些分派。这样的三维分派现在还不支持。

在工具箱中包含了一个名字为 mydwf 的例子。输入下面的代码，观察怎样使用。

首先输入：

```
help mydwf
```

得到下面的帮助信息：

```
MYDWF Example custom weight derivative function for MYWF.
Use this function as a template to write your own function.
Syntax
dZ_dP = mydwf('p',W,P,Z)
```

```
dZ_dW = mydwf('w',W,P,Z)
W - S×R weight matrix.
P - R×Q matrix of Q input (column) vectors.
Z - S×Q matrix of Q weighted input (column) vectors.
dZ_dP - S×R derivative dZ/dP.
dZ_dW - R×Q derivative dZ/dW.
Example
w = rand(1,5);
p = rand(5,1);
z = mywf(w,p)
dz_dp = mydwf('p',w,p,z)
dz_dw = mydwf('w',w,p,z)
$Revision: 1.1 $
```

接下来输入下面的代码：

```
w = rand(1,5)
p = rand(5,1)
z = mywf(w,p)
dz_dp = mydwf('p',w,p,z)
dz_dw = mydwf('w',w,p,z)
```

得到下面的结果：

```
w =
0.0056    0.2974    0.0492    0.6932    0.6501
p =
0.9830
0.5527
0.4001
0.1988
0.6252
z =
0.3856
dz_dp =
0.0110    0.3287    0.0393    0.2756    0.8129
dz_dw =
0.9663
0.3054
0.1601
0.0395
0.3909
```

使用下面的命令，可以显示出函数 mydtf 的实现。

```
type mydwf
```

得到函数 mydwf 的实现如下：

```
function d=mydwf(code,w,p,z)
%MYDWF Example custom weight derivative function for MYWF.
%
%    Use this function as a template to write your own function.
%
```

```
%    Syntax
%
%       dZ_dP = mydwf('p',W,P,Z)
%       dZ_dW = mydwf('w',W,P,Z)
%          W - S×R weight matrix.
%          P - R×Q matrix of Q input (column) vectors.
%          Z - S×Q matrix of Q weighted input (column) vectors.
%          dZ_dP - S×R derivative dZ/dP.
%          dZ_dW - R×Q derivative dZ/dW.
%
%    Example
%
%       w = rand(1,5);
%       p = rand(5,1);
%       z = mywf(w,p)
%       dz_dp = mydwf('p',w,p,z)
%       dz_dw = mydwf('w',w,p,z)
% $Revision: 1.1 $
% **   Replace the following calculations with your
% **   derivative calculation.   The only constraint is that
% **   the weight function must be a sum of elements, where
% **   each element i is a function of w(i) and p(i) only.
switch code
case 'p', d = 2*w.*p';
case 'w', d = p.^2;
otherwise, error(['Unrecognized code.'])
end
% **   Note that you have both the transfer functions input N and
% **   output A available, which can often allow a more efficient
% **   calculation of the derivative than with just N.
```

可以使用 mydwf 函数作为一个模板来生成自己的权重分派函数。

13.3.2 初始化函数

可以生成三种类型的初始化函数：网络、层，以及权重/偏置初始化函数，下面分别进行介绍。

1．网络初始化函数

最常规的一种网络初始化函数将所有的权重和偏置设置为一个值，这个值是训练或者自调整的一个合适的起始点。

函数定义以后，就可以将这个网络初始化函数指定到某个网络，如下所示。

```
net.initFcn = 'yournif';
```

这样，任何时候初始化网络时，这个网络初始化函数都会被使用。

```
net = init(net)
```

作为一个合法的网络初始化函数，函数必须接受并返回一个网络，如下所示：

```
net = yournif(net)
```

函数可以用任意的方法来设置网络的权重和偏置值。然而，要注意的是，不要改变其他的属性，或者把权重矩阵或偏置向量设置成错误的大小。出于性能方面的考虑，在调用初始化函数之前，init 关闭了对于网络属性的通常的类型检验。所以，如果把一个权重矩阵设置了错误的大小，将不会立即产生错误，但是在后面进行仿真或者训练网络时，将会带来问题。

如果感兴趣的话，可以在生成自己的网络初始化函数时，参考一下工具箱中函数 initlay 的实现。

2．层初始化函数

层初始化函数将某层所有的权重和偏置设置为一个适当的值，作为训练或自调整的起始点。

函数定义以后，就可以将这个层初始化函数指定到某个网络一层上。例如，下面的代码指定了层初始化函数 yourlif 到网络的第二层上。

```
net.layers{2}.initFcn = 'yourlif';
```

如果网络初始化函数（net.initFcn）被设置为工具箱函数 initlay，那么层初始化函数只被调用来初始化一个层。如果是这种情况，那么任何时候用 init 初始化网络时，函数都被用于初始化相应的层。

```
net = init(net)
```

作为一个合法的初始化函数，它必须接受一个网络和一个层标号作为输入，并且在初始化第 i 层后返回网络。

```
net = yournif(net,i)
```

这样，函数可以根据您的需要将第 i 层的权重，以及偏置值进行任意设置。然而，需要注意的是，不要改变其他的属性，或者是把权矩阵和偏置向量设置成了错误的大小。

如果对生成层初始化函数感兴趣，可以参考工具箱函数 initwb，以及 initnw 的实现。

3．权重及偏置初始化函数

权重及偏置初始化函数把所有的权重，以及偏置值设置为一个适当的值，作为训练或者自调整的起始点。

函数定义以后，就可以将这个初始化函数指定到网络中任意的权重和偏置上。例如，下面的代码指定了权重及偏置初始化函数 yourwbif 到网络第二层的偏置，以及从第一个输入到第二层的权重上。

```
net.biase{2}.initFcn = 'yourwbif';
net.inputWeights{2,1}.initFcn = 'yourwbif';
```

如果网络初始化函数（net.initFcn）被设置为工具箱函数 initlay，并且层初始化函数（net.layers{i}.initFcn）被设置为工具箱函数 initwb，那么权重及偏置初始化函数只调用来初始化一层。如果是这种情况，那么任何时候用 init 初始化网络，这个函数都会调用来初始化权重和偏置。

```
net = init(net)
```

作为一个合法的权重及偏置初始化函数，它必须接受两个输入，一个是某层的神经元的数目 S，另一个是一个 R 行的两列矩阵 $\boldsymbol{PR}$，它定义了 R 个输入的最小值和最大值，这

个函数返回一个新的权重矩阵 **W**。

```
W = rands(S,PR)
```

这里：

- S 是该层的神经元数目。
- PR 是一个 R×2 的矩阵，它定义了 R 个输入的最小值和最大值。
- W 是一个新的 S×R 权重矩阵。

函数还需要生成一个新的偏置向量，如下所示。

```
b = rands(S)
```

这里：

- S 是该层的神经元数目。
- b 是一个新的 S×1 的偏置向量。

输入下面的代码：

```
help mywbif
```

得到下面的帮助信息：

```
MYWBIF Example custom weight and bias initialization function.
Use this function as a template to write your own function.
Syntax
W = rands(S,PR)
S  - number of neurons.
PR - Rx2 matrix of R input ranges.
W - SxR weight matrix.
b = rands(S)
S  - number of neurons.
b - Sx1 bias vector.
Example
W = mywbif(4,[0 1; -2 2])
b = mywbif(4,[1 1])
$Revision: 1.1 $
```

接着输入下面的代码：

```
W = mywbif(4,[0 1; -2 2])
b = mywbif(4,[1 1])
```

得到下面的结果：

```
W =
0.0733    0.0754
0.0376    0.0794
0.0010    0.0920
0.0420    0.0845
b =
0.0368
0.0621
0.0731
0.0194
```

使用下面的代码来得到 mywbif 函数的实现。

```
type mywbif
```

得到函数的实现如下：

```
function w = mywbif(s,pr)
%MYWBIF Example custom weight and bias initialization function.
%
%    Use this function as a template to write your own function.
%
%    Syntax
%
%       W = rands(S,PR)
%         S   - number of neurons.
%         PR - Rx2 matrix of R input ranges.
%         W - S×R weight matrix.
%
%       b = rands(S)
%         S   - number of neurons.
%         b - S×1 bias vector.
%
%    Example
%
%       W = mywbif(4,[0 1; -2 2])
%       b = mywbif(4,[1 1])
% $Revision: 1.1 $
if nargin < 1, error('Not enough input arguments'), end
if nargin == 1
w = rand(s,1)*0.2;   % <-- Replace with your own initial bias vector
else
r = size(pr,1);        % <-- Replace with your own initial weight matrix
w = rand(s,r)*0.1;
end
```

可以把 mywbif 函数作为一个模板来生成自己的权重及偏置初始化函数。

13.3.3 学习函数

可以生成四种类型的学习函数：训练函数、自调整函数、性能函数，以及权重/偏置学习函数。下面分别进行介绍。

1．训练函数

网络训练函数是一种通常的学习函数。训练函数重复地把一组输入向量应用到一个网络上，每次都更新网络，直到达到了某种停止准则。停止准则可能是最大的学习步数、最小的误差梯度或者是误差目标等。

一旦定义了网络训练函数，就可以将其指定到某个网络上。

```
Net.trainFcn = 'yourtf';
```

这样在以后的任何时候训练网络时，都将使用这个网络训练函数。

```
[net,tr] = train(net,P,T,Pi,Ai)
```

作为一个合法的训练函数，它必须输入并返回一个网络。

```
[net,tr] = yourtf(net,Pd,Tl,Ai,Q,TS,VV,TV)
```

定义的训练函数必须提供关于自身的信息，它使用下面的调用格式。

```
info = yourtf(code)
```

对于每一个下列的字符串代码，返回相应的信息：

- 'version'　返回神经网络工具箱版本。
- 'pdefaults'　返回一个默认的训练参数结构体。

当设置网络训练函数（net.trainFcn）为自己的函数时，网络训练参数（net.trainParam）自动地设置为相应的默认结构。这些值可以在训练之前进行改变（或者保持不变）。

自己的函数可以以设定的任意方式更新网络的权重，以及偏置值。然而，需要注意的是，不要改变其他的属性值，或者把权重矩阵或偏置向量的大小设错。由于对性能的考虑，在调用自己的训练函数之前，train 对网络属性关闭了一般的类型检验功能。所以，如果将权重矩阵设置了错误的维数，并不能马上产生错误，但是将会在对网络进行仿真或者自调整时带来问题。

如果对生成自己的训练函数比较感兴趣，可以参考工具箱中一些函数的具体实现，如 trainc 及 trainr。对于这些效用函数的帮助列出了它们的输入输出项。

现在来讲效用函数。如果参考如 trainc、traingd 和 trainlm 这样的训练函数，就会发现它们使用了一组效用函数，这可以在 nnet/nnutils 目录里找到。

由于这些函数未来可能会进行改变，所以一般的参考中都没有给出它们，但是如果需要的话，则可以使用它们，只是这样就可能使得对于未来版本的工具箱需要对自己的程序进行调整。可以使用它们的帮助信息来得到函数的输入输出项。

下面的两个函数对于生成一个新的训练记录，并且在达到最终训练步数时进行截断处理的情况下是有用的。

- newtr　带有任意数目可选域的新训练记录。
- cliptr　将训练记录限制到最终训练步数上。

下面是这两个函数的帮助信息：

newtr 的帮助信息：

```
NEWTR New training record with any number of optional fields.
Syntax
tr = newtr(epochs,'fieldname1','fieldname2',...)
tr = newtr([firstEpoch epochs],'fieldname1','fieldname2',...)
Warning!!
This function may be altered or removed in future
releases of the Neural Network Toolbox. We recommend
you do not write code which calls this function.
```

cliptr 的帮助信息：

```
CLIPTR Clip training record to the final number of epochs.
Syntax
tr = cliptr(tr,epochs)
Warning!!
This function may be altered or removed in future
```

releases of the Neural Network Toolbox. We recommend
you do not write code which calls this function.

下面的三个函数计算网络的前向信号、误差，以及往回的性能分派：

- calca 计算网络输出及其他信号。
- calcerr 计算矩阵或者细胞数组误差。
- calcgrad 计算偏置及权重性能梯度。

下面给出了这三个函数的帮助信息：

calca 的帮助信息：

```
CALCA Calculate network outputs and other signals.
Syntax
[Ac,N,LWZ,IWZ,BZ] = calca(net,Pd,Ai,Q,TS)
Description
This function calculates the outputs of each layer in
response to a networks delayed inputs and initial layer
delay conditions.
[Ac,N,LWZ,IWZ,BZ] = CALCA(NET,Pd,Ai,Q,TS) takes,
NET - Neural network.
Pd  - Delayed inputs.
Ai  - Initial layer delay conditions.
Q   - Concurrent size.
TS  - Time steps.
and returns,
Ac  - Combined layer outputs = [Ai, calculated layer outputs].
N   - Net inputs.
LWZ - Weighted layer outputs.
IWZ - Weighted inputs.
BZ  - Concurrent biases.
Examples
Here we create a linear network with a single input element
ranging from 0 to 1, three neurons, and a tap delay on the
input with taps at 0, 2, and 4 timesteps.  The network is
also given a recurrent connection from layer 1 to itself with
tap delays of [1 2].
net = newlin([0 1],3,[0 2 4]);
net.layerConnect(1,1) = 1;
net.layerWeights{1,1}.delays = [1 2];
Here is a single (Q = 1) input sequence P with 8 timesteps (TS = 8),
and the 4 initial input delay conditions Pi, combined inputs Pc,
and delayed inputs Pd.
P = {0 0.1 0.3 0.6 0.4 0.7 0.2 0.1};
Pi = {0.2 0.3 0.4 0.1};
Pc = [Pi P];
Pd = calcpd(net,8,1,Pc)
Here the two initial layer delay conditions for each of the
three neurons are defined:
```

```
Ai = {[0.5; 0.1; 0.2] [0.6; 0.5; 0.2]};
Here we calculate the network's combined outputs Ac, and other
signals described above..
[Ac,N,LWZ,IWZ,BZ] = calca(net,Pd,Ai,1,8)
```

calcerr 的帮助信息：

```
CALCERR Calculates matrix or cell array errors.
E = CALCERR(T,A)
T - MxN matrix.
A - MxN matrix.
Returns
D - MxN matrix A-B.
E = CALCERR(A,B)
T - MxN cell array of matrices A{i,j}.
A - MxN cell array of matrices B{i,j}.
Returns
D - MxN cell array of matrices A{i,j}-B{i,j}.
```

calcgrad 的帮助信息：

```
CALCGRAD Calculate bias and weight performance gradients.
Synopsis
[gB,gIW,gIW] = calcgrad(net,Q,PD,BZ,IWZ,LWZ,N,Ac,gE,TS)
Warning!!
This function may be altered or removed in future
releases of the Neural Network Toolbox. We recommend
you do not write code which calls this function.
```

下面的两个函数用单个向量获取并设置网络的权重及偏置值。能够将所有的这些可调节的参数作为一个单一向量来处理，这对于实现优化算法是非常有用的。

- getx 作为一个单一向量获取所有的网络权重及偏置值。
- setx 作为一个单一向量设置所有的网络权重及偏置值。

下面给出这两个函数的帮助信息。

getx 数的帮助信息如下：

```
GETX Get all network weight and bias values as a single vector.
Syntax
X = getx(net)
Description
This function gets a networks weight and biases as
a vector of values.
X = GETX(NET)
NET - Neural network.
X   - Vector of weight and bias values.
Examples
Here we create a network with a 2-element input, and one
layer of 3 neurons.
net = newff([0 1; -1 1],[3]);
We can get its weight and bias values as follows:
net.iw{1,1}
```

net.b{1}
We can get these values as a single vector as follows:
x = getx(net);
See also SETX, FORMX.

setx 数的帮助信息如下：

SETX Set all network weight and bias values with a single vector.
Syntax
net = setx(net,X)
Description
This function sets a networks weight and biases to
a vector of values.
NET = SETX(NET,X)
NET - Neural network.
X - Vector of weight and bias values.
Examples
Here we create a network with a 2-element input, and one
layer of 3 neurons.
net = newff([0 1; -1 1],[3]);
The network has six weights (3 neurons * 2 input elements)
and three biases (3 neurons) for a total of 9 weight and bias
values. We can set them to random values as follows:
net = setx(net,rand(9,1));
We can then view the weight and bias values as follows:
net.iw{1,1}
net.b{1}
See also GETX, FORMX.

下面的三个函数对于实现优化函数也是非常有用的。第一个计算网络所有的前向信号，包括误差及性能；第二个反传以作为单一向量得到性能的分派；第三个函数反传以得到性能的雅可比行列式。后面的函数在高级优化技术，如在 Levenberg-Marquardt 中使用。

- calcperf 计算网络输出、信号，以及性能。
- calcgx 作为单一向量计算权重及偏置性能梯度。
- calcjx 作为单一向量计算权重及偏置雅可比行列式。

2．自调整函数

另一种通用的学习函数是自调整函数。自调整函数在每一个输入时间阶段更新网络时仿真网络，而这在进行下一个输入的仿真前完成。

一旦定义了自己的自调整函数，就可以指定它到一个网络上。

```
net.adaptFcn = 'youraf';
```

这样在任何时候调用自调整网络时，将调用自己的自调整函数。

```
[net,Y,E,Pf,Af] = adapt(net,P,T,Pi,Ai)
```

作为一个合法的自调整函数，它必须输入并且输出一个网络。

```
[net,Ac,El] = youraf(net,Pd,Tl,Ai,Q,TS)
```

自己定义的自调整函数必须提供关于自身的信息，它使用下面的调用格式。

```
info = youraf(code)
```

对于每一个下列的字符串代码，返回相应的信息：

- 'version' 返回神经网络工具箱版本。
- 'pdefaults' 返回一个默认的自调整参数。

当设置网络自调整函数（net.adaptFcn）为自己的函数时，网络自调整参数（net.adaptParam）自动地设置为相应的默认结构。这些值可以在自调整之前进行改变（或者保持不变）。

自己的函数可以以设定的任意方式更新网络的权重，以及偏置值。然而，需要注意的是，不要改变其他的属性值，或者把权重矩阵或偏置向量的大小设错。出于对性能的考虑，在调用自己的自调整函数之前，adapt 对网络属性关闭了一般的类型检验功能。所以，如果将权重矩阵设置了错误的维数，并不能马上产生错误，但是将会在对网络进行仿真或者训练时带来问题。

如果对生成自己的自调整函数比较感兴趣，可以参考工具箱中一些函数的具体实现，如 trains。

现在来讲一下效用函数。如果参考工具箱中惟一的自调整函数 trains，就可以发现它使用了一组效用函数，这可以在 nnet/nnutils 目录里找到。这些函数的帮助信息列出了它们的输入，以及输出项。

由于这些函数未来可能会进行改变，所以一般的参考中都没有给出它们，但是如果需要的话，可以使用它们，只是这样就可能使得对于未来版本的工具箱需要对自己的程序进行调整。

下面的三个函数对于仿真一个网络，并计算其性能的分派是有用的。

- calca1 带有任意数目可选域的新训练记录。
- calce1 将训练记录限制到最终训练步数上。
- calcgrad 计算偏置及权重性能梯度。

3．性能函数

性能函数允许对网络行为进行分级。这对于很多算法都是有用的，如回传算法，它通过调解网络权重及偏置的操作来改善性能。

一旦定义了自己的性能函数，就可以将其指定到某个网络上。

```
net.performFcn = 'yourpf';
```

任何时候训练或者自调整网络时，自己的网络初始化函数都会被调用。

```
[net,tr] = train(net,P,T,Pi,Ai)
[net,Y,E,Pf,Af] = adapt(net,P,T,Pi,Ai)
```

作为一个合法的性能函数，自己的函数必须用下面的方式调用：

```
perf = yourpf(E,X,PP)
```

如果 E 是一个细胞数组，可以通过下面的方式将其转换成一个矩阵：

```
E = cell2mat(E)
```

另外，自己的函数必须能够按下面的方式调用：

```
perf = yourpf(E,net)
```

这里，如果需要的话，可以通过下面的方式得到 X 和 PP 的值：

```
X = getx(net);
PP = net.performParam;
```

自己的性能函数同样必须提供关于自身的信息，它使用下面的调用格式。

```
info = yourpf(code)
```

对于每一个下列的字符串代码，返回相应的信息：

- 'version' 返回神经网络工具箱版本。
- 'deriv' 返回相关分派函数的名字。
- 'pdefaults' 返回一个默认的自调整参数。

当设置网络性能函数（net.performFcn）为自己的函数时，网络的性能参数（net.perforParam）将自动地设置为相应的默认结构。这些值可以在训练或者自调整之前进行改变（或者保持不变）。

下面的代码给出了一个定值的网络性能函数的例子。首先参考帮助信息。

```
help mypf
```

得到帮助信息如下：

```
MYPF Example custom performance function.
Use this function as a template to write your own function.
Calculation Syntax
perf = mypf(E,X,PP)
E   - Matrix or cell array of error vector(s).
X   - Vector of all weight and bias values.
PP - Performance parameter.
perf = mypf(E,net)
Information Syntax
info = mytf(code) returns useful information for each CODE string:
'version' - Returns the Neural Network Toolbox version (3.0).
'deriv'    - Returns the name of the associated derivative function.
'output'   - Returns the output range.
'active'   - Returns the active input range.
Example
e = rand(4,5);
x = rand(12,1);
pp = mypf('pdefaults')
perf = mypf(e,x,pp)
$Revision: 1.1 $
```

输入下面的代码：

```
e = rand(4,5)
x = rand(12,1)
pp = mypf('pdefaults')
perf = mypf(e,x,pp)
```

得到下面的结果：

```
e =
0.4451    0.8462    0.8381    0.8318    0.3046
0.9318    0.5252    0.0196    0.5028    0.1897
0.4660    0.2026    0.6813    0.7095    0.1934
0.4186    0.6721    0.3795    0.4289    0.6822
x =
```

```
0.3028
0.5417
0.1509
0.6979
0.3784
0.8600
0.8537
0.5936
0.4966
0.8998
0.8216
0.6449
pp =
x: 1
y: 0.5000
perf =
0.8152
```

使用下面的代码可以得到函数 mypf 的具体实现：

```
type mypf
```

其具体实现为：

```
function perf = mypf(e,x,pp)
%MYPF Example custom performance function.
%
%    Use this function as a template to write your own function.
%
%    Calculation Syntax
%
%      perf = mypf(E,X,PP)
%        E   - Matrix or cell array of error vector(s).
%        X   - Vector of all weight and bias values.
%        PP - Performance parameter.
%
%      perf = mypf(E,net)
%
%    Information Syntax
%
%      info = mytf(code) returns useful information for each CODE string:
%        'version' - Returns the Neural Network Toolbox version (3.0).
%        'deriv'     - Returns the name of the associated derivative function.
%        'output'   - Returns the output range.
%        'active'   - Returns the active input range.
%
%    Example
%
%      e = rand(4,5);
%      x = rand(12,1);
```

```
%       pp = mypf('pdefaults')
%       perf = mypf(e,x,pp)
% $Revision: 1.1 $
if nargin < 1, error('Not enough arguments.'); end
if isstr(e)
switch (e)
case 'version'
        perf = 3.0;            % <-- Must be 3.0.
case 'deriv',
        perf = 'mydpf';        % <-- Replace with the name of your
                                   %       associated function or ''
case 'pdefaults'
        perf.x = 1;            % <-- Replace with the your own performance
        perf.y = 0.5;          % <-- parameter structure or null matrix [].
otherwise, error('Unrecognized code.')
end
else
if isa(e,'cell')
e = cell2mat(e);
end
% ** delete the first expression below if you don't use PP, and
% ** the second if you don't use the weight and bias vector X.
if nargin == 2
pp = x.performParam;
     x = getx(net);
end
% ** Replace the following calculation with your own
% ** measure of performance.
numErrors = prod(size(e));
numWeightsBiases = length(x);
perf =   sum(sum(abs(e))) * pp.x/numErrors + ...
sum(abs(x)) * pp.y/numWeightsBiases;
end
```

可以将这个函数作为一个模板，用于生成自己的性能函数。

4．权重及偏置学习函数

最特殊的一类学习函数是权重及偏置学习函数。这些函数与某些训练及自调整函数一起，用于在学习中更新单独的权重及偏置。

一旦定义了自己的权重及偏置学习函数，就可以指定它们到网络中任意的权重和偏置上。例如，下面的代码指定权重及偏置学习函数 yourwblf 到第二层的偏置，以及到从第一个输入到第二层的权重上。

```
net.biases{2}.learnFcn = 'yourwblf';
net.inputWeights{2,1}.learnFcn = 'yourwblf';
```

如果网络训练函数（net.trainFcn）设置为 trainb、trainc 或 trainr 或者是网络自调整函数（net.adaptFcn）设置为 trains，权重及偏置学习函数只用于更新权重及偏置，那么任何

时候用 train 或者 adapt 训练或者自调整网络时，自己的函数将用于更新权重及偏置。

```
[net,tr] = train(NET,P,T,Pi,Ai)
[net,Y,E,Pf,Af] = adapt(NET,P,T,Pi,Ai)
```

作为一个合法的权重及偏置学习函数，它必须能够这样调用：

```
[dW,LS] = yourwblf(W,P,Z,N,A,T.E,gW,gA,D,LP,LS)
```

自己的函数按照下面的方式调用以更新偏置向量。

```
[db,LS] = yourwblf(b,ones(1,Q),Z,N,A,T,E,gW.gA,D,LP,LS)
```

这里：

- S 是该层的神经元数目。
- b 是一个新的 S×1 的偏置向量。

自己的函数同样必须提供关于自身的信息，它使用下面的调用格式。

```
info = yourwblf(code)
```

对于每一个下列的字符串代码，返回相应的信息。

- 'version'　返回神经网络工具箱版本。
- 'deriv'　返回相关分派函数的名字。
- 'pdefaults'　返回一个默认的自调整参数。

输入下面的代码可以得到关于这个定制的权重及偏置学习函数的帮助信息：

```
help mywblf
```

得到帮助信息如下：

```
MYWBLF Example custom weight and bias learning function.
    Calculation Syntax
      [dW,LS] = mywblf(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)
      [db,LS] = mywblf(b,ones(1,Q),Z,N.A,T,E,gW,gA,D,LP,LS)
        W   - S×R weight matrix (or S×1 bias vector).
        P   - R×Q input vectors (or ones(1,Q)).
        Z   - S×Q weighted input vectors.
        N   - S×Q net input vectors.
        A   - S×Q output vectors.
        T   - S×Q layer target vectors.
        E   - S×Q layer error vectors.
        gW - S×R gradient with respect to performance.
        gA - S×Q output gradient with respect to performance.
        D   - S×S neuron distances.
        LP - Learning parameters, none, LP = [].
        LS - Learning state, initially should be = [].
        dW - S×R weight (or bias) change matrix.
    Information Syntax
      info = mywblf(code) returns useful information for each CODE string:
        'version'      - Returns the Neural Network Toolbox version (3.0).
        'pdefaults' - Returns the name of the associated derivative function.
        'needg'        - Returns the output range.
    Example
      W = rand(4,5);
      gW = rand(4,5);
```

```
        lp = mywblf('pdefaults')
        [dW,ls] = mywblf(w,[],[],[],[],[],[],gW,[],[],lp,[]);
        W = W + dW;
        gW = rand(4,5);
        [dW,ls] = mywblf(w,[],[],[],[],[],[],gW,[],[],lp,ls);
        W = W + dW;
    $Revision: 1.2 $
```

输入下面的代码可以得到函数 mywbif 的具体实现：

```
type mywblf
```

得到函数 mywbif 的具体实现如下：

```
function [dw,ls] = mywblf(w,p,z,n,a,t,e,gW,gA,d,lp,ls)
%MYWBLF Example custom weight and bias learning function.
%
%   Calculation Syntax
%
%     [dW,LS] = mywblf(W,P,Z,N,A,T,E,gW,gA,D,LP,LS)
%     [db,LS] = mywblf(b,ones(1,Q),Z,N,A,T,E,gW,gA,D,LP,LS)
%       W  - S×R weight matrix (or S×1 bias vector).
%       P  - R×Q input vectors (or ones(1,Q)).
%       Z  - S×Q weighted input vectors.
%       N  - S×Q net input vectors.
%       A  - S×Q output vectors.
%       T  - S×Q layer target vectors.
%       E  - S×Q layer error vectors.
%       gW - S×R gradient with respect to performance.
%       gA - S×Q output gradient with respect to performance.
%       D  - S×S neuron distances.
%       LP - Learning parameters, none, LP = [].
%       LS - Learning state, initially should be = [].
%       dW - S×R weight (or bias) change matrix.
%
%   Information Syntax
%
%     info = mywblf(code) returns useful information for each CODE string:
%       'version'   - Returns the Neural Network Toolbox version (3.0).
%       'pdefaults' - Returns the name of the associated derivative function.
%       'needg'     - Returns the output range.
%
%   Example
%
%     W = rand(4,5);
%     gW = rand(4,5);
%     lp = mywblf('pdefaults')
%     [dW,ls] = mywblf(w,[],[],[],[],[],[],gW,[],[],lp,[]);
%     W = W + dW;
%     gW = rand(4,5);
```

```
%       [dW,ls] = mywblf(w,[],[],[],[],[],[],gW,[],[],lp,ls);
%       W = W + dW;
% $Revision: 1.2 $
if isstr(w)
switch lower(w)
case 'version'
        dw = 3.0;           % <-- Must be 3.0.
     case 'pdefaults'
        dw.lr = 0.01;     % <-- Replace with your own learning
                            %       parameters or the null matrix [].
     case 'needg'
        dw = 1;             % <-- 1 or 0 depending on whether your
                            %       function uses gW or gA, or not.
     otherwise
        error('Unrecognized property.')
end
else
if isempty(ls)
ls.x = 0.3;          % <-- Replace with your own functions initial
                            %       learning state or the null matrix []
end
dw = lp.lr*ls.x*gW; % <-- Replace with your own weight change
%       calculation.
ls.x = 1-ls.x;        % <-- Replace with your own learning state
%       update code, if you have any such state.
end
```

可以把函数 mywblf 作为一个模板，用于生成自己的权重及偏置学习函数。

13.3.4 自组织映射函数

有两种类型的函数用于控制神经元怎样自组织映射响应，它们是拓扑函数和距离函数。

1．拓扑函数

拓扑函数在给定维数的情况下，计算某层神经元的位置。

一旦定义了自己的拓扑函数，就可以指定它到网络的任意层上。例如，下面的代码指定拓扑函数 yourtopf 到网络的第二层。

```
net.layers{2}.topologyFcn = 'yourtopf';
```

任何时候当网络训练或自调整时，自己的拓扑函数将被使用。

```
[net,tr] = train(NET,P,T,Pi,Ai)
[net,Y,E,Pf,Af] = adapt(NET,P,T,Pi,Ai)
```

作为一个合法的拓扑函数，它必须从维数 dim 计算出位置 pos，如下所示：

```
pos = yourtopf(dim1,dim2,...,dimN)
```

工具箱中包含了一个定制拓扑函数的例子，名字为 mytopf。首先输入下面的的代码

得到帮助信息。

```
help mytopf
```

得到下面的帮助信息：

```
MYTOPF Example custom topology function.
Use this function as a template to write your own function.
Syntax
pos = mytopf(dim1,dim2,...,dimN)
dimi - number of neurons along the ith layer dimension
pos   - NxS matrix of S position vectors, where S is the
total number of neurons which is defined by the
product dim1*dim1*...*dimN.
Example
pos = mytopf(20,20);
plotsom(pos)
$Revision: 1.1 $
```

输入下面的代码：

```
pos = mytopf(20,20);
plotsom(pos)
```

得到的结果如图 13-8 所示。

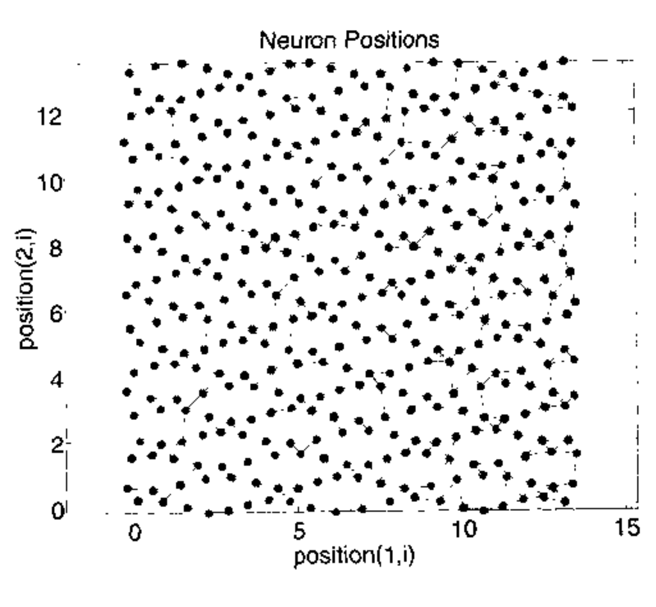


图 13-8 拓扑函数

输入下面的代码可以得到函数 mytopf 的实现。

```
type mytopf
```

可以得到 mytopf 的具体实现如下：

```
function pos=mytopf(varargin)
%MYTOPF Example custom topology function.
%
%     Use this function as a template to write your own function.
```

```
%
%   Syntax
%
%     pos = mytopf(dim1,dim2,....dimN)
%       dimi - number of neurons along the ith layer dimension
%       pos  - NxS matrix of S position vectors, where S is the
%              total number of neurons which is defined by the
%               product dim1*dim1*...*dimN.
%
%   Example
%
%     pos = mytopf(20,20);
%     plotsom(pos)
% $Revision: 1.1 $

% ** Replace the code below with your own calculation
% ** for the neuron positions.
dim = [varargin{:}];      % The dimensions as a row vector
size = prod(dim);          % Total number of neurons
dims = length(dim);        % Number of dimensions
pos = zeros(dims,size); % The size that POS will need to be
len = 1;
pos(1,1) = 0;
for i=1:length(dim)
dimi = dim(i);
newlen = len*dimi;
pos(1:(i-1),1:newlen) = pos(1:(i-1),rem(0:(newlen-1),len)+1);
posi = 0:(dimi-1);
pos(i,1:newlen) = posi(floor((0:(newlen-1))/len)+1);
len = newlen;
end
for i=1:2
pos(i,:)=pos(i,:)*0.7+sin([1:size]*exp(1)/5*i)*0.2;
end
```

可以将函数 mytopf 作为一个模板，使用它生成自己的拓扑函数。

2．距离函数

距离函数在给定位置的情况下，计算某一层中神经元的距离。

一旦定义了自己的距离函数，就可以指定它到网络的任意一层上。例如，下面的例子指定了距离函数 yourdistf 到网络的第二层。

```
net.layers{2}.distanceFcn = 'yourdistf';
```

这样，任何时候网络训练或者自调整时，自己的距离函数将会被使用。

```
[net,tr] = train(NET,P,T,Pi,Ai)
[net,Y,E,Pf,Af] = adapt(NET,P,T,Pi,Ai)
```

作为一个合法的距离函数，其必须从位置 pos 信息计算出距离 d，如下所示：

```
d = yourdistf(pos1,pos2,...,posN)
```

工具箱中包含了一个定制距离函数的例子，名字为 mydistf。首先输入下面的的代码得到帮助信息。

```
help mydistf
```

得到帮助信息如下：

```
MYDISTF Example custom distance function.
Use this function as a template to write your own function.
Syntax
d = mydistf(pos)
pos - NxS matrix of S neuron position vectors.
d    - S×S matrix of neuron distances.
Example
pos = gridtop(3,2);
d = mydistf(pos)
$Revision: 1.2 $
```

输入下面的代码：

```
pos = gridtop(4,5);
d = mydistf(pos)
```

得到结果如下：

```
d =
0        1.0000   2.0000   1.0000   1.5874   2.4473
1.0000        0   1.0000   1.5874   1.0000   1.5874
2.0000   1.0000        0   2.4473   1.5874   1.0000
1.0000   1.5874   2.4473        0   1.0000   2.0000
1.5874   1.0000   1.5874   1.0000        0   1.0000
2.4473   1.5874   1.0000   2.0000   1.0000        0
```

输入下面的代码，可以得到函数 mydistf 的具体实现：

```
type mydistf
```

得到函数 mydistf 的具体实现如下：

```
function d = mydistf(pos)
%MYDISTF Example custom distance function.
%
%    Use this function as a template to write your own function.
%
%       Syntax
%
%       d = mydistf(pos)
%          pos - NxS matrix of S neuron position vectors.
%          d    - S×S matrix of neuron distances.
%
%    Example
%
%       pos = gridtop(3,2);
%       d = mydistf(pos)
% $Revision: 1.2 $
s = size(pos,2);
```

```
for i=1:s
for j=1:s
% ** Replace the following line of code with your own
% ** measure of distance.
d(i,j) = norm(pos(:,i)-pos(:,j),1.5);
end
end
```

可以将函数 mydistf 作为一个模板，使用它生成自己的距离函数。

附录 A　MATLAB 6.5 的其他新特性

A.1　SIMULINK 5.0 的新特性

SIMULINK 是一种框图模型建模环境，用于动态系统的仿真、性能评估，并进行控制系统、DSP 系统和通信系统的设计。该软件在 GUI 和运行引擎方面的改进有：

- 形化的程序调试器，使错误分析及调试操作简单起来
- 支持大多数 SIMULINK 模块间以矩阵形式交换数据
- 支持更高速的基于帧信号处理的 DSP 应用
- 支持大模型开发的新特性，包括更新的的工具栏选项和增强的行编辑功能
- 集成的查找对话框工具方便了元件在 SIMULINK 模型和（或）状态流图中的搜索
- 增加了 SIMULINK 数据对象，支持用户自定义数据结构
- 新的航空库及示例
- 提高了库浏览器和模型浏览器的使用性能
- 废除了库连接，以方便库模型块的编辑
- 增强了的可配置子系统，便于设计模型向实际实现转化
- 具有筛选算法的高级查表模块，使仿真和代码生成更加快速、精确、灵活
- 单一视窗模式节省了宝贵的屏幕空间

A.2　MathWorks Release 13 新产品

1．航空工具箱（Aerospace Blockset）

航空工具箱以仿真为基础，提供具体的工具用于飞机、飞船、导弹，以及推进系统与子系统的建模、集成和仿真。

- 对于宇航式飞行器，提供了推进、控制系统、系统动力学，以及激励控制箱
- 用航空工具箱的组件提供了六自由度和三自由度的实例模型
- 包含重力、空气和风的环境模型
- 使用空间表示转换模块增强了坐标转换功能
- 能实现物理特性的单元转换
- 使用 Handle Graphics 技术，提供了动画模块，实现三自由度和六自由度的动画演示

2．曲线拟合工具箱（Curve Fitting Toolbox）

曲线拟合工具箱提供了数据预处理的各种方法，比如生成、比较、分析和管理模型。所有的功能既可以以命令行来实现，也可以以图形用户界面来实现。

- 预处理方法，包括数据的例、截断、滤波和奇异值的去处等
- 扩展了线形和非线形参数拟合模型库；对于非线形模型，起始点和参数的解算程序都进行了优化
- 有各种各样的参数和非参数模型，包括最小二乘法、加权最小二乘法和鲁棒拟合过程（包括有参数边界限制和没有参数边界限制）
- 自定义参数和非参数模型扩展
- 用样条和内插法实现非参数拟合
- 拟合数据的插补、外推、微风和积分

3．Motorola MPC555 嵌入式目标（Embedded Target for Motorola MPC555）

Motorola MPC555 嵌入式目标可以将由 MATLAB 实时工作空间编码器生成的代码直接配置到 MPC555 微控制器。Motorola MPC555 嵌入式目标依靠 MATLAB 实时工作空间编码器来为 Motorola MPC555 生成和提供具体的代码。

- 为 MPC555 提供快速原型和产品代码生成
- 针对 Motorola MPC555 MIOS, QADC 和 TouCAN 设备驱动块
- 通过板载调试器（onboard debugger）或 CAN（controller area network）下载应用到 Motorola MPC555
- 通过处理器在回路联合仿真的方式验证算法代码
- 生成详细的 HTML 代码生成报告，包括 RAM/ROM 信息

4．C6000 DSP 平台嵌入式目标（Embedded Target for C6000 DSP Platform）

简化 Texas Instruments DSP 软件设计和分析流程，直接从 MathWorks 环境生成高效代码。您可以使用 SIMULINK，DSP Blockset 和 Communications Blockset 中的块开发层次化的 DSP 算法框图模型，然后通过 Real-Time Workshop 生成无歧义的可执行算法，并可以进一步由 DSP 软件工程师优化。

- 自动生成支持 Texas Instruments DSP 目标的代码
- 无需 DSP 编程，就可以实时验证算法
- 生成 Code Composer Studio 项目格式的带注解的 C 代码

5．MATLAB COM Builder

MATLAB COM Builder 使得很容易将 MATLAB 算法转化为 COM 对象，可以用于任何基于 COM 的应用。

- 通过简便易用的 GUI 将 MATLAB 算法转换成 COM 对象
- 生成 COM 对象，可以供 Visual Basic, C/C++, Microsoft Excel 和其他任何基于 COM 的应用使用
- 允许修改和查看 MATLAB 生成的代码，可以对您所引用的函数有更好的理解
- 通过 MATLAB 生成的 COM 对象可以自由发布

6．MATLAB Excel Builder

MATLAB Excel Builder 使得很容易将复杂的 MATLAB 算法转化为独立的 Excel 插件（add-ins），用户可以利用以矩阵为基础、灵活的 MATLAB 编程环境，以及拥有大量可

用数学和图像函数的优点，从而迅速建立计算更深入的模型。

- 通过简便易用的 GUI 将 MATLAB 算法转换成 Excel 插件（add-in）
- 自动生成 .dll 和 Visual Basic 应用文件，并且可以输入到 Excel 中
- 生成的插件函数比 Visual BASIC for Applications (VBA)生成的快 95%
- 允许修改和查看 MATLAB 生成的代码，可以对您所引用的函数背后的逻辑有更好的理解

7．Code Composer Studio 开发工具的 MATLAB 连接

简化 Texas Instruments DSP 软件设计和分析流程，实现在 TI 软件开发环境、实时 DSP 硬件和 MATLAB 之间的通信。算法开发人员、系统设计人员和嵌入工程师通过此工具可以在 MATLAB 中测试、验证和可视化 DSP 软件，消除了 DSP 算法研究和实现之间的缺口。

- 提供 M 文件功能，允许您对数据传输和验证任务进行定制和自动化
- 在 MATLAB 和 Texas Instruments DSP 设备之间提供数据传输能力，而不必停止目标应用上执行的任务
- 硬件对象和 Code Composer Studio 帮助测试、验证和可视化 DSP 代码

8．基于模型校准工具箱（Model-Based Calibration Toolbox）

基于模型校准工具箱提供设计工具来校准功率系统。它建立在 MATLAB 高性能的计算环境，以及仿真能力的基础上，减少了功率器的检测时间，提高了工程效率，节省了校准时间，因而有潜力改进功率系统性能和可靠性。

- 新的经典设计 Plackett-Burman 两水平筛选设计（tow-level screening design），规则单纯形（Regular Simplex）——一个效率高的一阶设计方法
- 使用更方便，可以将用户信息存储起来，可以在 Model Browser 中跟踪项目文件的历史列表
- 新的上下文菜单，可以复制、删除和重命名模型
- 在现有的所有模型绘图放大功能之上，增加了新的数据绘图放大工具
- 在 CAGE 用户界面中浏览大型数据集，速度可以提高 40% 以上
- 支持 SIMULINK 库中的高级查表类型

9．机械仿真（SimMechanics）

它允许工程师在 SIMULINK 环境中仿真机械系统。应用 SimMechanics 可以直接建立机械部件的模型、仿真它们的运动，以及分析结果，而不需要推导数学方程。

- 增强的非线性方程求解器，仿真速度更快
- 优化的模型线性化
- 新的邻接（adjoining）特性，方便对相邻的其他物体的坐标系位置和方向的定义
- 对于并联机械模型，用户可以选择和观察切断铰链（cut joints）
- 图形用户界面增强，模型动化功能增强

具体内容读者可到网站 http://www.mathworks.com 查阅。



附录 B　MATLAB 6.5 安装问题指南

B.1　MATLAB 6.5 为什么安装后不能启动

经常有朋友遇到这个问题，其实在 Windows 98 或其第 2 版下安装 MATLAB 6.5 应该没有问题（也有部分可能会在安装了 IE5.5 或更高版本、更新了 Windows 的一些字库以后出现），而在 Windows ME 或 Windows 2000 下安装 MATLAB 6.5 不能启动的主要原因是它的部分中文字库和操作系统的字库重名造成冲突，下面给出几种可行的解决方法以供选择。

1. 更改区域设置

在控制面板里有区域设置一项，一般中文的 Windows 操作系统上默认的区域设置是“中文（中国）”，如图 B-1 所示。这里我们只要把它更改为“英语（美国）”之后重新启动计算机，MATLAB 6.5 就可以正常地运行了。不用担心，改过区域设置之后，操作系统仍然是中文操作系统，并不会给我们的使用带来太多的麻烦（当然有些小的显示等方面的问题）。这也是最省事的一种解决方法了。

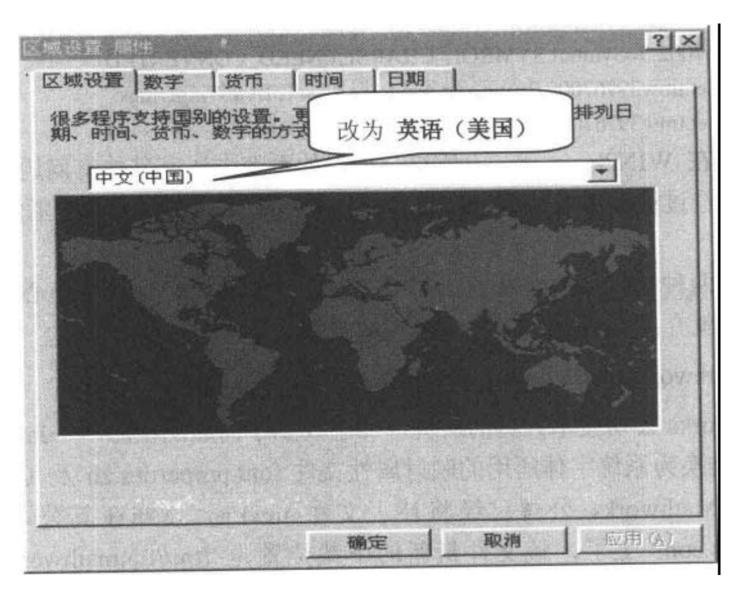


图 B-1 更改区域设置

2. 消除重名字库

可以直接在操作系统中将重名字库给删除，或是在 MATLAB 6.5 的字体配置文件中改

变重名字库相关信息。前一种删除方法将导致该字库从操作系统中被完全删除，其他应用软件将也无法使用该字库，因此并不十分理想，不过却也非常方便。后一种方法，该字库没有从操作系统中删除，只是在 MATLAB 6.5 的字体属性配置文件中改变了相关信息，因此并不影响其他应用软件使用这些字库。这两种删除方法共同的最核心的问题就在于要知道是哪一些字库重名了，而这往往需要去试验或是获得他人的经验。目前在一般的情况下，大约只有"新宋体"这个字库有重名的问题。第一种删除字体的方法也非常简单（在字体目录下将相应的文件删除即可），但一般不提倡使用。这里，我们就介绍如何更改 MATLAB 6.5 的字体配置文件。

首先，MATLAB 6.5 这个导致死机的字体属性配置文件是 font.properties.zh，这个文件在目录$MATLAB\sys\java\jre\win32\jre\lib\下（$MATLAB 表示 MATLAB 6.5 的安装目录)。我们可以用一个文本编辑器来编辑这个文件。

在 $MATLAB\sys\java\jre\win32\jre\lib\font.properties.zh 文件的最后加上：

```
# Font entry added by The MathWorks:"新宋体"
tmw36525210.0=\u65b0\u5b8b\u4f53,ANSI_CHARSET
tmw36525210.1=\u65b0\u5b8b\u4f53,GB2312_CHARSET
tmw36525210.2=WingDings,SYMBOL_CHARSET,NEED_CONVERTED
tmw36525210.3=Symbol,SYMBOL_CHARSET,NEED_CONVERTED
fontcharset.tmw36525210.2=sun.awt.windows.CharToByteWingDings
fontcharset.tmw36525210.3=sun.awt.CharToByteSymbol

# Font entry added by The MathWorks:"@新宋体"
tmw39767002.0=@\u65b0\u5b8b\u4f53,ANSI_CHARSET
tmw39767002.1=@\u65b0\u5b8b\u4f53,GB2312_CHARSET
tmw39767002.2=WingDings,SYMBOL_CHARSET,NEED_CONVERTED
tmw39767002.3=Symbol,SYMBOL_CHARSET,NEED_CONVERTED
fontcharset.tmw39767002.2=sun.awt.windows.CharToByteWingDings
fontcharset.tmw39767002.3=sun.awt.CharToByteSymbol
```

以上结果为在 WIN2k sc pro 上所得，如果的系统中还有其他有问题的中文字库，需要同样地处理。经过检验可以找到一些没有问题的中文字库如：宋体、黑体、仿宋、楷体、幼圆和隶书。

这种方法的两种手段都比较麻烦，需要一些经验和试验，如果不小心还可能会出一些问题，下面我们就介绍 Mathworks 公司的解决方案。

3．使用 Mathworks 公司提供的技术支援

经过 Mathworks 公司技术人员的研究，终于找到了问题的根源——Java 虚拟机用来将标准 Java 字体转换为系统字体所用的映射属性文件 font.properties.zh 及其映射程序中存在一个小 BUG。Mathworks 公司已经将补丁文件 mwt.jar 发布在其公司的网站 http://www.mathworks.com 上了，该文件最新的下载位置是 ftp://ftp.mathworks.com/pub/tech-support/solutions/s26990。下载了该文件后将其替换目录$MATLAB\java\jar\下的同名文件（最好对初始的文件做一个备份)。然后，重新启动 MATLAB 6.5，问题就可以解决了，如果仍有一些其他问题，可以到其公司或其他一些相关网站上查询。

B.2　安装时更新 Java 虚拟机的问题

这个问题一般比较偶然地出现在 Windows 98 和 Windows NT 环境中，有时因为安装其他软件时已经解决了这个问题，在安装 MATLAB 6.5 时就会比较顺利。但有时确实会出现要求更新 Java 虚拟机的提示消息框，这时我们应当更新我们的 Java 环境。其软件一般可以在 MATLAB 6.5 的安装盘中找到，如果是下载的软件，可以再到网址 http://www.microsoft.com/java/vm/dl_vm40.htm 去下载相应的软件 msvm.exe（对应于 Windows 98 和 Windows NT 操作系统）。如果是 Windows 2000 操作系统，只需要安装升级补丁 SP2。

B.3　PDF 文档的获取

国内的 MATLAB 6.5 软件大多是下载的软件，因此帮助文件不全。下面的这两个网址可以给大家一点帮助。

国外：http://www.mathworks.com/access/helpdesk/help/helpdesk.shtml

国内：http://science.fire.ustc.edu.cn/matlab6.html

附录C　MATLAB神经网络工具箱函数参考

MATLAB 神经网络工具箱提供了丰富的函数，在第 2 章中对其中的主要函数进行了详细说明，本附录按类别列出了神经网络工具箱提供的函数，并对其进行简单描述。

C.1　工具箱函数

工具箱函数主要如表 C-1～C-26 所示。

表 C-1　网络类型函数

函数名称	功　能
assoclr	相关学习规则
backprop	BP 网络
elman	Elman 回归网络
hopfield	Hopfield 回归网络
linnet	线性网络
lvq	学习向量量化
percept	感知器
radbasis	径向基网络
selforg	自组织网络

表 C-2　分析函数

函数名称	功　能
errsurf	单输入神经元表面误差
maxlinlr	线性神经元最大学习速率

表 C-3　距离函数

函数名称	功　能
boxdist	两个位置向量间的距离
dist	欧氏距离权重函数
linkdist	连接距离函数
mandist	曼哈顿距离权重函数

表 C-4 图形接口函数

函数名称	功能
nntool	神经网络工具箱图形用户接口

表 C-5 层初始化函数

函数名称	功能
initnw	Nguyen-Widrow 层初始化函数
initwb	权重及偏置层初始化函数

表 C-6 学习函数

函数名称	功能
learncon	公平偏置学习函数
learngd	梯度下降权重学习函数
learngdm	梯度下降动量权重学习函数
learnh	Hebb 权重学习函数
learnhd	退化的 Hebb 学习规则
learnis	不固定权重学习函数
Learnk	Kohonen 权重学习函数
learnlv1	LVQ1 权重学习函数
learnlv2	LVQ2 权重学习函数
learnos	Outstar 权重学习函数
learnp	感知器权重及偏置学习函数
learnpn	规范化感知器权重及偏置学习函数
learnsom	自组织映射权重学习函数
learnwh	Widow-Hoff 权重及偏置学习规则

表 C-7 线性搜索函数

函数名称	功能
srchbac	使用回溯搜索的一维最小化
srchbre	使用 Brent 方法的一维间隔定位
srchcha	使用 Charalambous 方法的一维最小化
srchgol	使用黄金分割搜索的一维最小化
srchhyb	使用二分/立方混合搜索的一维最小化

表 C-8 网络输入分派函数

函数名称	功能
dnetprod	乘积网络输入分派函数
dnetsum	求和网络输入分派函数

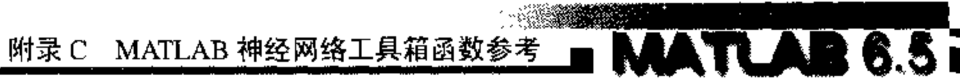


表 C-9 网络输入函数

函数名称	功 能
netprod	乘积网络输入函数
netsum	求和网络输入函数

表 C-10 网络初始化函数

函数名称	功 能
initlay	逐层网络初始化函数

表 C-11 网络应用函数

函数名称	功 能
adapt	允许神经网络自适应
disp	显示神经网络的属性
display	显示神经网络变量名及属性
init	初始化神经网络
sim	仿真神经网络
train	训练神经网络

表 C-12 生成新网络函数

函数名称	功 能
network	生成一个通常的神经网络
newc	生成一个竞争层
newcf	生成一个前向层叠 BP 网络
newelm	生成一个 Elman BP 网络
newff	生成一个前馈 BP 网络
newfftd	生成一个前馈输入延时 BP 网络
newgrnn	设计一个泛化的回归网络
newhop	生成一个 Hopfield 回归网络
newlin	生成一个线性网络
newlind	设计一个线性网络
newlvq	生成一个学习向量量化网络
newp	生成一个感知器
newpnn	设计一个概率神经网络
newrb	设计一个径向基网络
newrbe	设计一个精确的径向基网络
newsom	生成一个自组织映射网络

表 C-13　性能分派函数

函数名称	功　能
dmae	平均绝对误差性能分派函数
dmse	均方差性能分派函数
dmsereg	均方差 w/reg 性能分派函数
dsse	误差平方和性能分派函数

表 C-14　性能函数

函数名称	功　能
mae	平均绝对误差性能函数
mse	均方差性能函数
msereg	均方差 w/reg 性能函数
sse	误差平方和性能函数

表 C-15　绘图函数

函数名称	功　能
hintonw	权矩阵 Hinton 图
hintonwb	权矩阵及偏置向量 Hinton 图
plotbr	贝叶斯调整训练网络性能绘制
plotep	在误差表面绘制权重及偏置位置
plotes	绘制单输入神经元误差表面
plotpc	在感知器向量图上绘制分类线
plotperf	绘制网络性能
plotpv	绘制感知器输入目标向量
plotsom	绘制自组织映射
plotv	将向量绘制为从原点发出的直线
plotvec	用不同的颜色绘制向量

表 C-16　预处理及后期处理函数

函数名称	功　能
postmnmx	把被 premnmx 规范化的数据反规范化
postreg	后期处理网络响应 w 的线性化回归分析
poststd	把被 prestd 规范化的数据反规范化
premnmx	规范化数据到-1 和 1 之间
prepca	对输入数据进行主成分分析
prestd	规范化数据为 0 均值单位标准差
tramnmx	带预计算最小和最大的传递函数

（续表）

函数名称	功　　能
trapca	带通过 prepca 计算的 PCA 矩阵的传递函数
trastd	预计算均值和标准差的传递函数

表 C-17　仿真支持函数

函数名称	功　　能
gensim	为神经网络仿真生成一个仿真模块

表 C-18　拓扑函数

函数名称	功　　能
gridtop	Gridtop 层拓扑函数
hextop	Hexogonal 层拓扑函数
randtop	随机层拓扑函数

表 C-19　训练函数

函数名称	功　　能
trainb	用权重及偏执学习规则成批训练
trainbfg	BFGS 类牛顿回传
trainbr	贝叶斯规范化
trainc	循环循序递增更新
traincgb	Powell-Beale 连接梯度回传
traincgf	Fletcher-Powell 连接梯度回传
traincgp	Polak-Ribiere 连接梯度回传
traingd	梯度递减回传
traingda	带自调整 lr 回传的梯度递减
traingdm	带动量回传的梯度递减
traingdx	带动量及自调整 lr 回传的梯度递减
trainlm	Levenberg-Marquardt 回传
trainoss	一步正切回传
trainr	随即顺序递增更新
trainrp	带反弹的回传（Rprop）
trains	顺序递增更新
trainscg	量化连接梯度回传

表 C-20　传递分派函数

函数名称	功　　能
dhardlim	硬限幅传递分派函数
dhardlms	对称硬限幅传递分派函数
dlogsig	对数 S 型传递分派函数
dposlin	正线性传递分派函数
dpurelin	线件传递分派函数
dradbas	径向基传递分派函数
dsatlin	饱和线件传递分派函数
dsatlins	对称饱和线性传递分派函数
dtansig	双曲线正切 S 型传递分派函数
dtribas	三角基传递分派函数

表 C-21　传递函数

函数名称	功　　能
compet	竞争传递函数
hardlim	硬限幅传递函数
hardlims	对称硬限幅传递函数
logsig	对数 S 型传递函数
poslin	正线性传递分派函数
purelin	线性传递函数
radbas	径向基传递函数
satlin	饱和线性传递函数
satlins	对称饱和线件传递函数
softmax	软最大传递函数
tansig	双曲线正切 S 型传递函数
tribas	三角基传递函数

表 C-22　效用函数

函数名称	功　　能
calca	计算网络输出及其他信号
calca1	对单个时间步长计算网络信号
calce	计算层误差
calce1	对单个时间步长计算层误差
calcgx	作为单个向量计算权重及偏置性能梯度
calcjejj	计算雅可比性能向量

（续表）

函数名称	功能
calcjx	作为单个矩阵计算权重及偏置性能雅可比矩阵
calcpd	计算延时的网络输入
calcperf	计算网络输出、信号以及性能
formx	组合偏置及权重到单个向量中
getx	作为单个向量获得网络权重及偏置的值
setx	作为单个向量设置网络权重及偏置的值

表 C-23 向量函数

函数名称	功能
cell2mat	组合一个细胞数组矩阵到一个矩阵中
combvec	生成所有的向量组合
con2seq	把并行向量转换成串行向量
concur	生成并行偏置向量
ind2vec	把标号转换成向量
mat2cell	把矩阵拆成细胞数组矩阵
minmax	矩阵行的范围
normc	规范化矩阵的列
normr	规范化矩阵的行
pnormc	伪规范化矩阵的列
quant	离散化值作为一个量的倍数
seq2con	把序列向量转换成并行向量
sumsqr	一个矩阵的元素平行和
vec2ind	把向量转换成标号

表 C-24 权重及偏置初始化函数

函数名称	功能
initcon	公平偏置初始化函数
initzero	零权重及偏置初始化函数
midpoint	中点权重初始化函数
randnc	规范化列权重初始化函数
randnr	规范化行权重初始化函数
rands	对称随机权重/偏置初始化函数
revert	改变网络权重和偏置为最近的初始化值



表 C-25 权重分派函数

函数名称	功　能
ddotprod	点积权重分派函数

表 C-26 权重函数

函数名称	功　能
dist	欧氏距离权重函数
dotprod	点积权重函数
mandist	曼哈顿距离权重函数
negdist	负距离权重函数
normprod	规范化点积权重函数

C.2 传递函数图形

传递函数图形如图 C-1～图 C-12 所示。

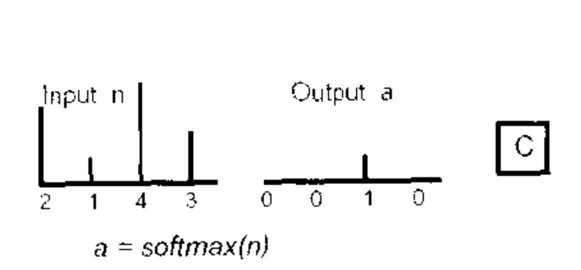


图 C-1 竞争传递函数

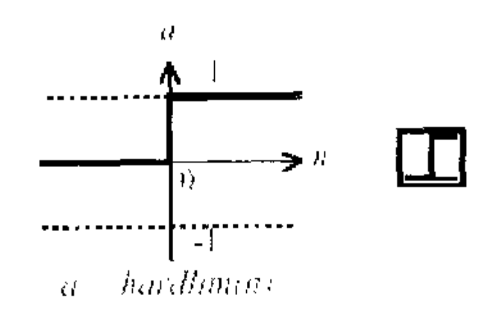


图 C-2 硬限幅传递函数

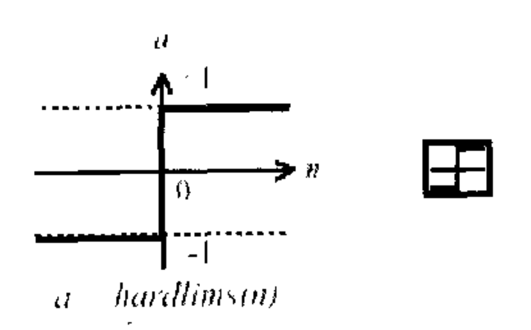


图 C-3 对称硬限幅传递函数

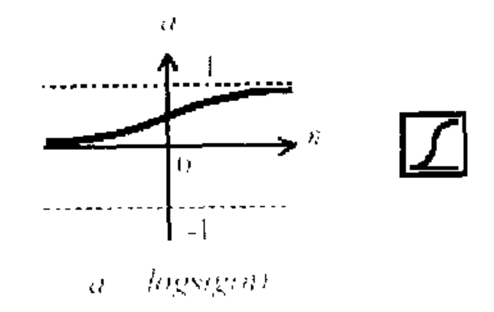


图 C-4 log-s 型传递函数

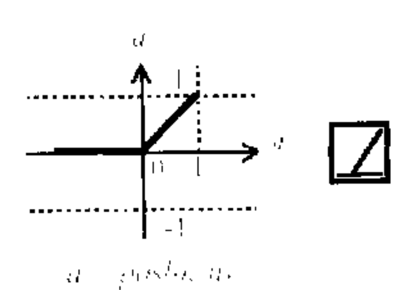


图 C-5　正线性传递函数

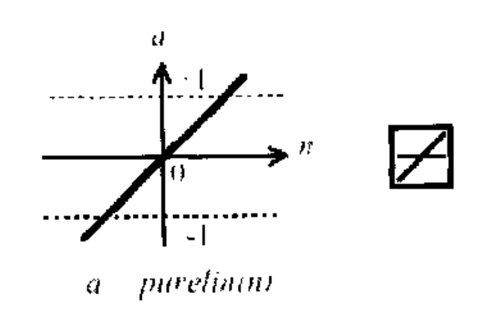


图 C-6　线性传递函数

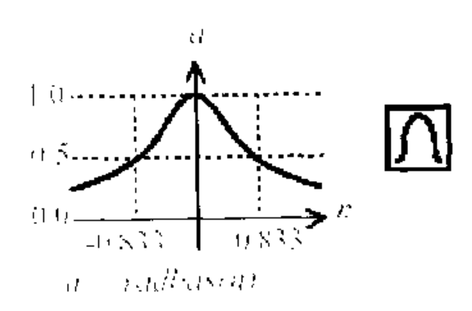


图 C-7　径向基函数

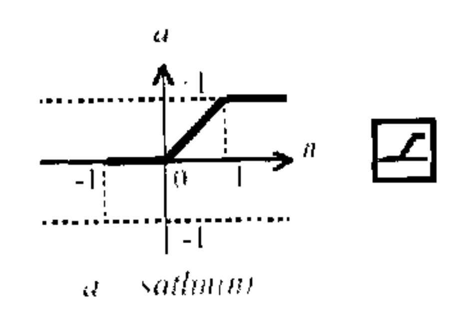


图 C-8　Satlin 传递函数

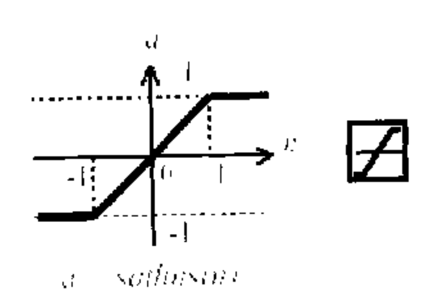


图 C-9　Satlins 传递函数

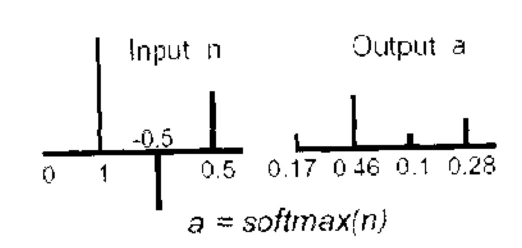


图 C-10　软最大传递函数

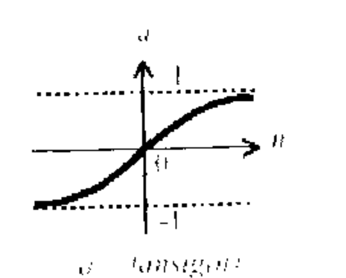


图 C-11　tan-s 型传递函数

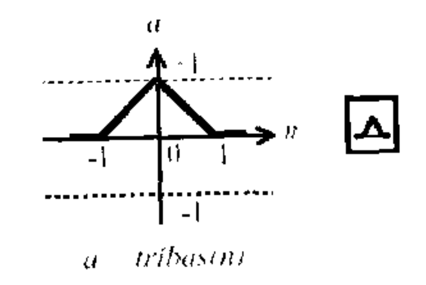


图 C-12　三角基函数

参 考 文 献

1 胡守仁等编著. 神经网络导论. 长沙: 国防科技大学出版社, 1993
2 Neural Network Toolbox. MathWorks, 2000
3 袁曾任编著. 人工神经元网络及其应用. 北京: 清华大学出版社
4 楼顺天等编著. 基于 MATLAB 的系统分析与设计——神经网络. 西安: 西安电子科技大学出版社, 1998
5 Simon Haykin. Neural Networks: a Comprehensive Foundation[second edition]
6 The MathWorks, Inc. http://www.mathworks.com,2002